PowerPC Reference Platform Specification
Version 1.1

Document Editor: Dr. Mark Dean
Document Editor: Arthur Adkins

11400 Burnet Rd.
Austin, TX 78758

Original Signed by Dr. Mark Dean, October 14, 1994
Director, Power Personal Systems Architecture

Document Created: October 14, 1994
Next Revision/Review Date: July 1, 1995

THIS DOCUMENT IS AT AN APPROVED VERSION 1.1 LEVEL

The responsibility for using the latest version of this document lies with the user of the document. To verify that you have the latest version, contact either Motorola or IBM. Also, verify that your copy is not older than the next revision date. If a new version is available, it should be used and this version should be discarded.

LICENSE INFORMATION

Copyright: To the extent that IBM has a right in the *PowerPC Reference Platform Specification* (including accompanying source code samples), IBM authorizes you to copy and distribute this publication (including accompanying source code samples) in any form, without payment to IBM, for the purpose of developing original publications, code or equipment (except integrated circuit processors) which conform to this publication (including accompanying source code samples) and for the purpose of using, reproducing, marketing, and distributing such original publications, code or equipment (except integrated circuit processors). This authorization applies to the content of this specification only and not to the referenced material.

In consideration you agree to include on each reproduction of any portion of these publications (including accompanying source code samples) or any derivative works based thereon that are marketed or distributed to others a copyright notice as follows: "(C) Copyright (your company name), (year). All rights reserved."

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization. If you fail to comply with the above terms, your authorization terminates.

Patents: IBM and others may have patents or pending patent applications or other intellectual property rights covering the subject matter described herein. This document neither grants or implies a license or immunity under any IBM or third party patents, patent applications or other intellectual property rights other than as expressly provided in the above copyright license. IBM assumes no responsibility for any infringement of third party rights resulting from your use of the subject matter disclosed in, or from the manufacturing, use, lease or sale of products described in, this document.

Licenses under IBM's utility patents in the field of information handling systems are available on reasonable and non-discriminatory terms. IBM does not grant licenses to its appearance design patents. Direct your licensing inquiries in writing to the IBM Director of Licensing, International Business Machines Corporation, 208 Harbor Drive - MS#7, Stamford, CT 06904.

Third Edition - Version 1.1 (October 1994)

NOTICES

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law. In such countries, the minimum country warranties will apply.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THE DISCLAIMER OF WARRANTY APPLIES NOT ONLY TO THE PUBLICATION (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) BUT ALSO TO ANY COMBINATIONS, INCORPORATIONS, OR OTHER USES OF THE PUBLICATION (INCLUDING ACCOMPANYING SOURCE CODE EXAMPLES) UPON WHICH A CLAIM COULD BE BASED.

Some states do not allow disclaimers of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

These materials could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in, or accompanying, this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such reference or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for copies of this publication or for technical information about IBM products described herein should be directed to IBM Microelectronics or to Motorola.

It is not the intention of the *PowerPC Reference Platform Specification* to address all system environments which use PowerPC microprocessors. Some environments (such as high-end technical and commercial servers, PDAs or set top boxes) may find other reference specifications more appropriate. Contact other standards bodies for system environments not covered by this *PowerPC Reference Platform Specification*.

TRADEMARKS AND SERVICE MARKS

Trademarks or service marks of the IBM Corporation in the United States or other countries are denoted by an asterisk (*) on their first occurrence in this publication.

Trademarks or service marks of other corporations are denoted by a double asterisk (**) on their first occurrence in this publication. See the section entitled "Trademark Information" for a complete listing of trademarks and the companies that own them.

Preface

This version of the *PowerPC Reference Platform Specification* includes changes and enhancements which have been suggested by reviewers within the team which is developing this specification. In addition, valuable suggestions and experience have been obtained by interacting with users of this specification and from participants in classes on this subject. This information has been incorporated into this version of the specification.

| The editors acknowledge the contributions of John Osman, Keith Diefendorff, Steve Johns, Susumu
| Shimotono, Haruo Sugi, Charles Barbour, Glen Miranker, and Spencer Worley, who developed some of this material. Rich Oehler and Ron Hochsprung provided overall review and contributed along with Luan
/ Nguyen, Mitch Bradley, and David Kahn to the Open Firmware material. Sections of this document were developed using design documents developed by Rich Bealkowski, Ralph Begun, Sunny Lam, Frank Levine,
/ Eliseo Pena, Jim Peterson, Randy Swanberg, Howard Tanner, Ken Uplinger, Koichi Kii, Barry Wolford,
| and Wendel Voigt. Portions of the software appendices were contributed by Walt Daniels, Bob Stephens,
/ Ken Borgendale, Catherine Wildermuth, Jim Mott, Steven Zucker, Abe Ellenberg, John O'Quin, Conrad
/ Sloat, and Bob Willcox. Phil Gerskovich and Judy Chavis have contributed by promoting the specification.
/ Bill Hufler and Andrew Lucchesi handled the publication details. Brad Frey, Ray Pedersen, Sean Curry,
/ Steve Thurber, and Kumar Ranganathan have provided comments and information.

/ Linda Buckley's and David Tjon's departments, including Don McCauley, Ken Nordhauser, Shien-Tai Pan, Yongjae Rim, Allan Steel, Gary Tsao, and Furong Zhang, have helped immensely by contributing to sections based on their experience providing training to users of this specification.

| Peter Bahrs, Ram Gupta and Ramon Pantin have contributed to this specification by reviewing the docu-
| ment and representing IBM software in the workgroup for the PowerPC processor binding to IEEE 1275.

Contents

Figures

Tables

Summary of Changes

Table 1 lists changes for each version of this document from the initial Alpha Release in November 1993. Changes which are clerical are not listed. Within the text, changes which have been made for the current Version 1.1 are marked with a "|" symbol. Changes which were made for Version 1.0 are marked with a "/" symbol.

Table 1 (Page 1 of 9). History of Changes to the Specification

Content of the Change	Reason for the Change
Changes for the Beta Version of the specification	
Added Workplace OS material to the appendix and summarized it in the System Configuration section.	This material was not available for the Alpha release.
Changed processor designation to "PowerPC 601."	This is a vendor-neutral designation.
Changed heading and initial paragraph for Section 3.17, "PowerPC Architecture Features Not Recommended."	To clarify that these were system recommendations and not processor recommendations.
Redrew portable diagram to match implementations.	To create a family resemblance in implementations.
Added AIX memory map restrictions to the appendix and the memory map architecture section.	AIX requires reserved memory.
Changed the hardware configuration section -- used the term "alphanumeric device" instead of keyboard.	This is an implementation-independent term.
Showed recommendation for parity and ECC memory.	This subject was not addressed.
Changed "main memory" to "system memory" in the multiprocessor appendix.	To make consist with the definitions in the memory map section.
Made modifications of the I/O Master view of the Memory Map figure.	There was an error in the location of Flash ROM.
Changed title of Section 2.9, "Industry Interface Standards," and defined keyboard interfaces as examples.	List of keyboard interfaces was not complete.
Identified specific IDE standard.	To provide definition of IDE standard.
LocalTalk added to Table 4 as recommended.	LocalTalk needed on other configurations.
Added Ethernet adaptor and removed "ISA."	Section on Compliant Subsystem and Devices does not show IBM 16-bit and Ethernet adaptors.
Serial ports 3 and 4 added to Table 14	Serial ports 3 and 4 were not covered in Reference Implementation.
Modified the Workstation and Portable reference implementations.	To explain the number of sync instructions in the PowerPC 601 Endian switch example and to show an example for other processors.

Table 1 (Page 2 of 9). History of Changes to the Specification	
Content of the Change	**Reason for the Change**
Section deleted.	Firmware development information is not architecture and is too product-specific.
Spelled out MCA on first use and put in acronym table.	To differentiate from Music Corporation of America.
Added statement to require an abstraction.	To clarify that the software must protect self modified code from loss of coherency.
Referenced the different PCI buses in the figure for the technical workstation description in the appendix.	The description for a technical workstation's reference implementation needs to clarify that there are two PCI buses.
Some clarification added.	Some of the features in the section on Power Management Hardware Features needed to be clarified.
Clarified the effect on hardware and software if the operating system does not have an abstraction layer accessible to third parties.	Clarified the second note in the System Abstraction Layer section.
Clarified that direct store segments support is an option.	Previously unclear whether direct store support was required.
Clarified that cache inhibited string operations to System Memory do not need to be supported.	Inconsistency in the description of handling string operations.
Wording changes and references to the PowerPC architecture in the Hardware Configuration section.	Requirements are not clearly defined in the Hardware Configuration section.
Changed "should" to "recommended."	To make terminology consistent in the configuration and architecture sections.
Clarified the alignment requirements.	The alignment requirements reference an implementation and the rationale needs to be defined.
Made changes to Reference Implementation to address what was implemented and eliminated architectural alternatives.	Reference Implementation gave too many architectural alternatives.
Changed figure and description for ISA I/O master access to memory.	Description and figure for I/O masters did not match.
Included an energy-managed workstation in the appendix and deleted some equipment from the portable description.	The portable configuration described in the appendix contained components not found on portables.
Described the Bi-Modal graphics adaptor approach in the Endian appendix.	Need to describe Bi-Modal graphics adaptors.
Improved descriptions of AIX.	AIX appendix did not define the scope and version.
Edited for clarity in the Upgrade Slot Definition.	Some of the L2 cache protocol is not clear.
Inserted the map of the CMOS RAM in the Reference Implementation.	The map of CMOS RAM was not included.
Added new figures and text in the Machine Abstraction section.	To clarify the abstraction software requirements.

Table 1 (Page 3 of 9). History of Changes to the Specification	
Content of the Change	**Reason for the Change**
The tables in the operating system appendices and in the Hardware Configuration section were changed to remove VGA and allow 640-by-480 resolution.	To clarify graphics requirements for operating systems' hardware.
Added stand-alone diagnostics to "Configuration and Diagnostics."	Single user workstations may not need on-line diagnostics.
Audio subsystems in reference platform and portable implementation example now use Crystal Semiconductor CS4231 audio integrated circuit.	Design change.
Added description of Scatter/Gather DMA.	Clarification.
Deleted references to VGA Hardware Emulation.	VGA Hardware Emulation components were deleted from the design.
Clarified upgrade slot information.	Upgrade slot does not support multiprocessing.
Added an implementation example for MP Interrupts.	Additional level of detail.
Rewrote most of the Power Management section in architecture.	Rewrite was due to results of PM review meetings.
Provided details of support for suspend and hibernate states.	Clarification.
VPD section in Firmware section was moved to residual data section.	VPD is defined in residual data structure.
Defined NVRAM structure.	Definition of NVRAM was required.
Re-wrote Architecture Compliance Testing.	Added four phases of testing. Added section on branding.
Tagged index items and created index in the back of the document.	Index needed.
Changed "60x" to "PowerPC" or "PowerPC processor."	"60x" does not take into account 620 or future products.
Changed "CPU bus," "60x bus," "local bus" to "primary processor bus" or "PowerPC processor bus."	"60x" incorrect; consistency needed.
Changed RTC to NVRTC in Section 6.	NVRTC more accurate and not confused with 601 RTC.
Changed "conflict" to "corrupt" and "NT" to "Windows NT" in Section 6.	Clarification.
Changed "memory access" to "system memory address" in Section 6.	To ensure consistency and avoid confusion.
Removed "h" from hex nomenclature and changed "cache line" to "buffer line" in Section 6.	For consistency and clarification.
Deleted rows from System I/O Address Map table pertaining to "VGA Emulation."	Changed to reflect current I/O Address Map.

Table 1 (Page 4 of 9). History of Changes to the Specification	
Content of the Change	**Reason for the Change**
Created acronym table.	To define acronyms for readers unfamiliar with them.
Created glossary.	To define terms and make specification usable to a diverse audience.
Added new information to Multiprocessor section in appendix.	More detail regarding symmetric multiprocessor systems needed.
Portable System section in Appendix A replaced with updated information.	For clarification and additional information.
Title of Section 3 changed to “Architecture Guidance.”	“Guidance” clarifies intent of section.
New language inserted in Section 1.1 regarding interfaces.	Illustrates that key features of PowerPC Reference Platform architecture include use of industry standard interfaces.
New language added to Section 3.1 regarding system memory coherency.	Previously omitted.
Language in Appendix C regarding operating systems removed.	Necessary correction.
GTX Graphics Subsystem added to PCI Adapter list in Appendix C.	Previously omitted.
Rearranged information regarding multiprocessor interrupts in Section 3.	This information was extensive enough to warrant its own subsection.
Added information to multiprocessor section regarding in-line L2 caches.	To explain the need for in-line L2 caches in SMP systems.
/ Changes for Version 1.0	
/ Updated Appendix F and Hardware Configuration / sections to clarify the requirements to support / Workplace OS.	Some of the Workplace OS requirements were not clear.
/ Reorganized and reworded the subsystem / description in Section 2.	Requirements were not crisply defined in the Hardware Configuration section.
/ Changed to a software-controlled setting of frame / buffer and graphics registers in Section 2.4.4	Independent setting of the frame buffer and graphics registers is not required.
/ Clarified information regarding hardware / coherency in Section 2.2.4.	I/O memory on the processor bus may not always participate in the hardware coherency.
/ Changed the table of contents to just chapters and / major sections within a chapter.	Table of contents was too detailed.
/ Several wording changes such as “directly / attached” changed to “directly attached or / built-in” in the Hardware Configuration section.	Some wording in the Hardware Configuration section does not consider portables.
/ Added a recommendation to provide scrolling / capability.	Portable display devices may not have resolution as high as the recommended resolution.
/ Described the branding and certification in the / Introduction.	Verification is not architecture.

Table 1 (Page 5 of 9). History of Changes to the Specification	
Content of the Change	**Reason for the Change**
Reformatted chapter 2 and 3 to separate requirements.	Chapter 2 and 3 did not clearly define requirements.
Clarified the time base in the processors.	Time base runs at a slower frequency than the processor bus.
Defined system requirements of I/O access from the processor.	Bus bridge requirements were not clear.
Added warning that some operating systems do not maintain coherency.	AIX expects hardware-maintained coherency.
Increased the minimum hardfile size to 120 MB.	Hardfile size was too small for any operating system.
Clarified hardfile implementation information.	“Hardfile” may imply only spinning media.
Defined hardfile size as formatted, uncompressed space.	Hardfile size was not specific.
Media detection requirement added.	Some operating systems use media detection.
Indicated ISO 9660 required software support.	ISO 9660 does not effect the hardware configuration
Recommended wiring the CD-ROM directly to the audio subsystem.	Reference Implementation designed this way.
Clarified pointing device requirements.	Pointing device requirements were not clear.
Clarified and expanded audio requirements.	Audio requirements were not precise and need more description.
Added Bi-Endian graphics adaptor requirement.	Big-Endian applications which write directly to graphics need Big-Endian adaptors.
Made parallel port optional.	A smaller footprint is possible without a parallel port.
Clarified DMA controller information.	DMA controllers should not alias.
Recommended the enhanced IDE.	Enhanced IDE will have several advantages.
Added and updated references in Section 2.9.8.	PCI and PCMCIA references needed to be updated.
Added PCMCIA information in Section 2.9.9.	The PCMCIA requirements needed to be expanded.
Added Solaris information to Chapter 2 and Appendix G.	Solaris information was missing.
Explained the software impact of storage combining in Section 3.2.	The architecture allows storage combining.
Defined the code sequence for synchronizing the I and D cache in Section 3.5.	It was not clear why an operating system service is needed.
Added information regarding Bi-Endian design in Section 3.12.	The impacts on software of Bi-Endian design were not completely defined.
Defined terms and corrected errors in the tables in Section 3.15 and 3.16.	It was not clear what combinations of size and address the system hardware must support.

Table 1 (Page 6 of 9). History of Changes to the Specification	
Content of the Change	**Reason for the Change**
Included the device drivers in the RTAS definition in Section 4.4.	Device drivers provide an abstraction function.
Abstraction only required in Section 4.4.3, if the operating system manages coherence.	Flushing buffers is not supported by some operating systems.
Provided for alternate sources of PCMCIA Socket Services.	Some operating systems supply PCMCIA Socket Services device drivers.
Added new appendix section.	To provide information about the PowerPC Binding to IEEE 1275.
Added discussion of implementation attributes of coherence.	Subtleties of maintaining coherence needed to be explained in Section 3.2.
Noted that the 601 prefetches as if storage was not guarded.	Section 3.3 required clarification that the 601 treats storage as not guarded.
Defined operation in an environment with load and store combining.	Load and store combining has an effect on System I/O, Section 3.4.
Clarified the approach for bus resource locking.	Section 3.6 needed to define bus resource locking.
Section 3.8 -- defined addresses for 64-bit implementations.	A 64-bit address space may change addresses.
Section 3.8 now recommends an approach for abstracting the memory map.	Memory map is not defined in the architecture.
Indicated that software must program to the weakly ordered model.	Section 3.9 indicated that systems must be weakly ordered.
Noted changes in the approach for supporting hibernation and suspend.	Section 3.11.1 was not consistent with the current implementations.
Clarified the software and hardware support for power management.	Section 3.11.3 needed to be clarified.
Defined Bi-Endian requirements and implementation.	Section 3.12 must allow all Bi-Endian implementations.
Restructured Section 3.13.	Section 3.13 has some non-multiprocessor information and implementation information.
Defined the requirements for a bridge in Section 3.15 and added load and store combining considerations.	To define the specific requirements.
Defined the requirements for a bridge in Section 3.16 and corrected some entries in the table.	To define the specific requirements.
Clarified that non-word-aligned floating-point uncached loads or stores need not be supported.	Section 3.17.6 does not allow word-aligned floating-point.
Clarified that the special direct store should be encapsulated.	Section 3.17.8 must allow use of the special direct-store segment.
Added clarification in Section 4.1.	It was not clear that hardware vendors are responsible for supplying replacement abstraction.
Allowed for PCMCIA device drivers.	Section 4.4.12 did not discuss PCMCIA drivers.

Table 1 (Page 7 of 9). History of Changes to the Specification	
Content of the Change	**Reason for the Change**
Reworked Section 5.0 to include the Open Firmware requirement and describe both the conventional (i.e. legacy) approach and Open Firmware approach.	Open Firmware should be made a requirement.
Defined the addressing state in Section 5.4.1.	The addressing mode at the end of boot was not clear.
Added Section 5.4.2, which defines the conditions for getting service from Open Firmware during run-time.	Open Firmware may supply some function during run-time.
Added Section 5.5.5.2 to describe NVRAM.	The NVRAM header file is hard to read.
Section 5.6 notes that Open Firmware will replace Residual data.	Open Firmware supplies information which is also contained in Residual data.
Added section 5.6.2.	Some terminology in Residual data is Plug and Play.
Added section 5.7 to describe Open Firmware.	Open Firmware is in the appendix.
Defined serial port addresses and reserved some addresses.	Section 6.1.5 does not define all serial port addresses.
Made changes in Sections 6.1.5.3, 6.1.5.8, and 6.1.5.9.	The meaning of some bits changed in port 80C, 81C, and 850.
Section 6.1.6 defines that all PCI interrupts go to 15.	PCI interrupts are routed to a single interrupt.
Section 6.3.3.1, A.1.8, and A.2.5 were changed.	System I/O bridge changed to a ZB part number.
Section 6.4 was modified to show different Endian switching and clarify that this approach applies only to 601.	Endian switching with cache enabled is easier.
Section 6.4 defines the implications on boot and Endian switching.	Endian switching is dependent upon the location of the byte reversal.
Sections 6.7.1 and 6.7.4 were changed.	Some signal names were changed.
Section 6.7.3 includes better figures and more part numbers.	To define the SRAM used in the cache.
Section A.1.2 was modified to show different Endian switching and clarify that this approach applies to other PowerPC processors.	Endian switching with cache enabled is easier.
Section A.1.2 defines the implications of boot and Endian switching.	Endian switching is dependent upon the location of the byte reversal.
Changed section A.1.13.	PCMCIA controller was changed.
Section A.4 and A.5 show more PCI adaptors.	More PCI adaptors should be included in higher-performance systems.
Clarified and expanded the implementation information in Section A.6.	Alternate interrupt structures needed for multi-processors.
Section B.4 indicates multiple apertures are an alternative.	Multiple apertures could be used to address each graphics pixel depth.

Table 1 (Page 8 of 9). History of Changes to the Specification	
Content of the Change	**Reason for the Change**
Section C.1 included.	Some devices (e.g. keyboard) are listed under a bus heading.
Expanded list of graphics adaptors in Section C.2.	Latest models of graphics adaptors were not listed.
Added PCMCIA bridges in Section C.3.	To expand the list of PCMCIA bridges.
Added explanation in Section C.5.	The 1.2 MB capacity varies the disk speed.
Section D.1 now lists the Windows NT porting center.	A Windows NT porting center is available.
Clarified processors supported in Sections D.5.1, E.5.1, and F.5.1.	Not every PowerPC processor may be supported by every operating system.
Sections D.5.5, D.12, E.5.4, E.12, F.5.4, F.5.5, and F.12 were changed to indicate what was required by the standard hardware configuration.	Some requirements for devices are not operating system requirements.
Sections D.6, E.6, and F.6 were clarified.	Tape drive is recommended as a backup medium.
Section D.8 includes a new table.	To indicate the effect on software of hardware changes.
Section E.1 contains new contact numbers.	AIX contact numbers were incorrect.
Sections D.3, E.3, and F.3 were added.	To include some of the operating environment characteristics of the operating system.
Sections E.1 and E.4 were changed.	The AIX product plan was revamped.
Sections E.5.5 and E.5.6 were changed.	AIX is dependent upon the native attachment of devices.
Section E.8 -- removed any notion of a abstraction layer.	The AIX kernel is the software abstraction for AIX.
Reworded Section F.1.	Workplace OS is not an operating system, but a kernel which supports operating systems.
Appendix G was filled out.	Information for the Solaris operating system was obtained.
Appendix I now contains the PowerPC supplement to IEEE 1275 for Open Firmware.	To publish the PowerPC specific Open Firmware information.
The Bibliography was expanded to include every referenced document.	The Bibliography was not complete.
The Additional Information section was expanded.	To indicate where documents can be ordered.
Changes for Version 1.1	
Various clerical changes.	To correct some minor typographical errors.
Corrected references to *The PowerPC Architecture* manual in Chapter 2 and the Bibliography.	This manual is now published by an outside publisher.
Added recommendation for software in Sections 2.3.2 and 6.3.3.2.	The floppy light will light during polling.
Identified the IEEE definition of the ISA bus in Section 2.9.10.	The IEEE ISA bus document was not defined.

Table 1 (Page 9 of 9). History of Changes to the Specification	
Content of the Change	**Reason for the Change**
Clarified the resource locking information in Section 3.6.	The resource locking section needed more information.
Deleted reference in Section 3.12.1.	A doubleword byte reversal instruction was not defined.
Indicated a second source for the paper in Section 3.12.1.	To make it easier for the public to obtain technical reports.
Deleted a requirement in Section 5.1.2.	There is no reason to restrict booting to 16 colors.
Indicated in Section 5.4.1 that the boot is allowed to pass control in Little-Endian mode in some situations.	Boot should be able to pass control to Little-Endian operating systems in Little-Endian mode.
Changes made in Sections 5.5.5.1, 5.6.1 and 5.6.2.	Development has shown errors and a need for additional information in the NVRAM, Residual and PNP structures.
Defined the CRC algorithm in Section 5.5.5.2.	The CRC polynomial was not defined.
In Section 5.6, described and referenced the additional appendices.	To include vendor-specific Plug and Play extensions used by AIX and a residual data dump.
Clarified in Section 6.2.5 the clock chips used in the Reference Implementation.	To define the clock chips.
In Appendix A.1.14, defined the Western Digital device.	The graphics adaptor in the Portable was not defined.
Several changes were made in Appendix A.1.	The implementation of the portable has changed slightly.
Described AIX 4.1.1 in Appendix E.	AIX has been updated since this specification was first published.
Appendix I is the updated version of the PowerPC supplement.	The PowerPC Open Firmware committee has an approved version of the supplement.
The section titled “Obtaining Additional Information” was updated.	This specification is available electronically.

1.0 Introduction

Today's computer systems exist in a wide range of environments, from hand-held portables to room-size mainframes. The largest percentage of computer systems are personal-use systems based on the IBM PC/AT*, Apple Macintosh**, or a variety of workstation-level RISC architectures. These machines cover the needs of personal productivity, entry engineering design, entry commercial data management, information analysis, and database, file, and application servers. But with all their performance and functionality, their architectures can limit the systems designer's ability to add new features without jeopardizing operating system or application compatibility. These limitations restrict the use of hardware and software enhancements which promise improved user interfaces, faster system performance, and broader operating environments.

Technological advances have outpaced the computer industry's ability to utilize them. Application and
/ operating system compatibility issues prevent system designers from using many new architectures and interfaces. Many times, system designers must carry obsolete hardware structures to maintain compatibility. To be sustainable and continue to grow, the computer industry must define computer architectures which allow system and application designs to utilize the latest silicon, interface, storage, display and software technologies. The key to these new computer architectures is the ability of the software to abstract the hardware from the operating system kernel and applications without sacrificing compatibility or performance.

1.1 PowerPC Reference Platform Philosophy

The creators of the *PowerPC* Reference Platform Specification* believe that software, from power-on self test (POST) and diagnostics to operating systems and applications, drives the usability and acceptance of a computer system. The computer user judges the effectiveness of a system by its user environment, responsiveness, functionality, and reliability. The system software controls these attributes by leveraging the hardware features and performance to provide a total system solution.

Independent software vendors need the promise of a large installed base of hardware systems to justify the development expense for today's operating systems and applications. To create this large installed base of systems, an industry-standard computer system architecture is required. This computer system architecture must yield systems for the personal computer and workstation industry that leverage the latest digital technologies. The key features of the architecture must be: 1) its ability to allow hardware vendors to differentiate, 2) its ability to use industry-standard components and interfaces, and 3) its ability to support optimization of operating system and application performance. This type of open system architecture allows hardware system vendors to develop differentiated yet compatible systems -- each system is able to
/ run any of the compatible operating systems as well as applications ported to those operating systems and the system architecture.

1.2 Purpose of Document

The *PowerPC Reference Platform Specification* provides a description of the devices, interfaces, and data formats required to design and build a PowerPC based industry-standard computer system. It is written to create a hardware standard, which when coupled with the hardware abstraction software provided by the operating system or hardware system vendors, allows the computer industry to build PowerPC systems which all run the same shrink-wrapped operating systems and the same shrink-wrapped applications for those operating environments. This specification defines a system architecture which covers most traditional computer systems, from portables to servers. It gives system developers the freedom to choose the level of market differentiation and enhanced features required in a given computing environment without carrying obsolete interfaces or losing compatibility.

This specification defines the minimum functional requirements needed for a compliant PowerPC Reference Platform implementation. It also provides a list of recommended hardware subsystems, devices and interfaces, which if used in a PowerPC Reference Platform implementation, yield a level of functionality required by most operating environments. This specification also describes a reference implementation which is a fully functional PowerPC Reference Platform system design supporting all operating systems and applications which are being ported to this reference platform. This Reference Implementation provides an example to which system developers can compare, allowing them a better understanding of their own design goals. The reference implementation may be built by any system vendor seeking to minimize development expense for software and hardware. But hardware platform developers maintain freedom of implementation below the level of the abstracted interfaces defined by each of the operating systems. A hardware system vendor may change subsystems from those used in the Reference Implementation. Abstraction software must be supplied for the supported operating systems by either the hardware system vendor or operating system vendor.

The *PowerPC Reference Platform Specification* is written primarily for system developers. It defines the hardware devices, subsystems, interfaces and firmware required for a compliant implementation. It contains operating system-specific descriptions and references to their hardware abstraction approach. The combination of abstraction software and this pliant hardware specification allows system vendors to differentiate while maintaining binary compatibility at the operating system and application level. This system implementation flexibility is called “compatible differentiation.” Compatibility is achieved through the use of industry-standard interfaces, system structures, device drivers, and software abstraction of hardware subsystems. The software abstraction of hardware provides expanded opportunities for differentiation by allowing enhancement of the Reference Implementation with devices abstracted from the operating systems and applications. Since abstraction layers and device drivers are operating system dependent, this differentiation comes at a cost -- differentiation requires the hardware system vendor or operating system vendor to develop software for each operating system to be supported.

Subsystem and chip vendors may also reference these specifications when developing support devices for these systems. But many of these vendors will want to consult with a system design group to understand the requirements of a given environment. It will be impossible for a single chip set to provide optimal performance and function for all cost points and all operating environments. Cost, performance, functionality, silicon process characteristics and development time will drive many design decisions.

Finally, operating system vendors may use this specification as a reference to determine the level of functionality required in a hardware abstraction layer. The specification shows the hardware subsystems which are likely to change and therefore may need hardware abstractions. Recommendations regarding the functionality of abstraction layers are also made.

1.3 PowerPC Reference Platform Goals

The goals of this specification are as follows:

- To create an open industry standard to be used for the implementation of PowerPC based systems and to support the hosting of operating systems and applications to PowerPC Reference Platforms. This specification itself is available to the industry and can be used by any hardware or software vendor to develop PowerPC products.
- To provide a specification which covers most traditional computing environments, from portables to servers. Eventually, nontraditional information systems like PDAs and Personal Communicators will be covered as addenda or new chapters to the base specification. This specification will grow as new computing environments are defined. This specification will continue to provide open industry-standard system architectures to hardware and software vendors.
- To leverage the high-volume personal computer component market for chip sets, devices and subsystems whenever possible. Many parts of a computer system need not be differentiated from other competitors.

Certain system attributes like low-speed communication ports can be easily implemented with off-the-shelf parts. Being able to use readily available personal computer components minimizes system cost; minimizes development time and expense; provides multiple suppliers; and simplifies porting of many operating systems, firmware and device drivers.

- To leverage existing and future industry-standard buses and interfaces. Existing bus architectures (i.e. ISA, VME, Micro Channel Architecture* (MCA), NuBus, etc.) provide an established base of adaptors and are well understood by card and system designers. These existing bus architectures also have a proven level of performance and functionality. Also, established industry-standard interfaces (i.e. SCSI, IDE, LocalTalk**, Ethernet**, etc.) and newer bus architectures, interfaces and protocols (i.e. PCI, PCMCIA, Serial SCSI, ATM, etc.) provide higher levels of performance or utility not achievable by the older standards. The *PowerPC Reference Platform Specification*, coupled with software abstractions of hardware and device drivers, allows the system designer to determine which buses, interfaces, and protocols best suit the target system environment.
- To allow compatible differentiation through the use of abstracted hardware interfaces and device drivers. The operating systems written for PowerPC Reference Platform-compatible systems will provide a predefined level of loadable abstractions. This allows hardware subsystem and interface variations without affecting compatibility with the operating system kernels and their respective applications. The *PowerPC Reference Platform Specification* DOES NOT define a universal BIOS in ROM because that would tie all operating systems to a single difficult-to-change interface, defined in terms of current technology. Operating systems can optimize their abstraction interfaces while supporting a wide range of environments and implementations. This structure can also leverage new software and hardware technologies without losing compatibility with older systems or applications.
- To provide address map relocation. Another key attribute of this specification is the relocatability of devices and subsystems within the PowerPC address space. Subsystem address information, which defines where I/O devices reside, is stored by the system designer and passed to the operating systems. The architecture also allows the use of multiple and identical buses and adaptors in the same system without address conflicts. This is very important in computing environments requiring a significant amount of I/O.
- To place control of power management in the operating system. It is important that the combination of hardware and software systems be designed to minimize power consumption through automatic power-saving methods. For environmental and cost reasons, systems not being used should minimize their power consumption. The goal is to have all PowerPC Reference Platform systems be power-conscious and conserve energy whenever possible.

1.4 Scope

The *PowerPC Reference Platform Specification* is targeted primarily to system design houses, but provides valuable information for operating system, device driver, adaptor, and ASIC vendors. It will also assist value-added resellers. The specification supports all 32-bit PowerPC processors. It is intended to cover the following systems: portables, medialess systems, desktops, workstations, and servers. The specification allows support for multiple operating systems, each using different methods of abstracting hardware variations. Finally, because the *PowerPC Reference Platform Specification* requires machine abstractions, the specification accommodates the evolution of software and hardware technologies without losing system compatibility.

The *PowerPC Reference Platform Specification* covers these main areas:

- Hardware Configuration

  The hardware configuration defines the minimum and recommended hardware standards and capacities required to be PowerPC Reference Platform compliant and compatible with targeted operating environ-

ments. This section describes memory system, storage media, human interfaces, I/O device and expansion requirements for a PowerPC Reference Platform-compliant system.

- Architecture

  The system architecture defines the minimum and recommended hardware system attributes required to design a compatible computer system. This section describes the key hardware and software architecture attributes and restrictions defined for PowerPC Reference Platform compliance.

- Machine Abstractions

  The Machine Abstractions section defines in general the approaches that software should take to bridge differences within PowerPC Reference Platform subsystems. Specific implementations of the machine abstraction are described and referenced in appendices for each operating system. System vendors may use this material to decide which subsystems can vary between PowerPC Reference Platform implementations.

- Boot Process and Firmware

  This section provides information on standard software features supported by ROM-based system code. This covers all code executed before control is passed to the operating system kernel. Storage locations for product configuration data are also defined. This section defines the PowerPC Reference Platform boot architecture, which supports all targeted operating systems. This section also defines the boot structure used for loading operating systems from floppy, hardfile, CD-ROM, or networks.

- Reference Implementation

  This section describes an example implementation of a PowerPC Reference Platform-compliant system. This description may be used as a high-level design for vendors wanting to produce a compatible system or may be used as an example for vendors who want to produce a differentiated system. For those who require more detail, design kits for this reference implementation are available and may be obtained through the telephone numbers listed in the section of this document entitled “Obtaining Additional Information.”

1.5 PowerPC Reference Platform Brand and Certification

To support the establishment of the PowerPC Reference Platform as an open standard and to verify consistency with the architecture, a PowerPC Reference Platform brand and certification process will be established. A hardware platform which passes the hardware compliance verification may use the “PowerPC Reference Platform Compliant” brand or icon. Operating systems which pass the operating system compliance verification may also use this label. This brand will be a helpful communications tool within the marketplace, identifying systems which are ready to use compatible operating systems and applications. Also, this brand will be helpful within the development community, between system developers and operating system developers, for the communication of function and requirements.

One or more independent laboratories will be qualified to provide the *PowerPC Reference Platform Specification* certification for systems and operating systems. These laboratories will perform the verification and approve results to provide certification of a PowerPC Reference Platform-compliant hardware system or operating system.

For certification, a hardware system must demonstrate compliance to all hardware and firmware requirements in this specification in Sections 2.0, “Hardware Configuration,” 3.0, “Architecture Guidance,” and 5.0, “Boot Process and Firmware.” The system vendor will need to provide design details to the certification laboratory in enough detail to demonstrate compliance to the requirements. The system vendor will have to provide samples of the hardware system to the certification laboratory. The certification laboratory will perform inspections and run test software on these samples.

/ In addition, a hardware system must demonstrate that it runs one of the operating systems ported to a
/ PowerPC Reference Platform. These operating systems are listed in the appendices of this specification.
/ The system vendor must provide any requisite abstraction software to the certification laboratory. The labo-
/ ratory will use a suite of functional tests to demonstrate that an operating system and some of its applica-
/ tions run on the platform. Systems certified as PowerPC Reference Platform compliant will be permitted to
/ use the brand and must identify the specific operating systems which they support. However, all operating
/ systems do not need to be certified on a vendor's system.

/ Operating systems which have been ported to PowerPC Reference Platform-compliant systems and which
/ comply with the software requirements in this document may be certified as PowerPC Reference Platform
/ compliant. Operating system requirements occur primarily in Section 4.0, "Machine Abstractions," but
/ some functions are defined in other sections. An operating system vendor would present its design and the
/ PowerPC version of the operating system to the certification laboratory. The laboratory would verify
/ through inspection and testing that the operating system met the requirements. An operating system which
/ is certified may display the PowerPC Reference Platform brand. Those that do not comply with the require-
/ ments may identify the specific hardware systems to which they have been ported.

/ Applications which have been rehosted to run on PowerPC Reference Platforms may be labeled as
/ "PowerPC Reference Platform Ready" and indicate the operating system under which the application runs.
/ No certification of applications is planned.

2.0 Hardware Configuration

/ This section describes standard subsystems that make up PowerPC Reference Platform-compliant systems.
/ The minimum compliant configuration and several other configurations are specified as sets of these subsys-
/ tems. Configuring PowerPC Reference Platform systems from standard subsystems guarantees a set of func-
/ tions and capabilities that are available to the operating systems and application software.

/ The next eight subsections discuss functions, requirements and recommendations pertaining to the processor,
/ memory, storage, human interface, Real-Time Clock, connectivity, expansion bus(es), and additional subsys-
/ tems. The ninth subsection defines interface standards for some of those subsystems. The last subsection
/ defines five typical configurations and summarizes required and recommended subsystems for each configura-
/ tion as well as for a minimum PowerPC Reference Platform-compliant machine. A table at the end of the
/ section shows the minimum requirements for a subset of each operating system targeted to be hosted on
/ PowerPC Reference Platform hardware systems.

/ Some capabilities described within this section are required to support at least one of the operating systems. These capabilities are stated in terms of "must." Every PowerPC Reference Platform-compliant machine must have these capabilities. Some capabilities are recommended for better usability or performance or to allow all operating systems to run on a PowerPC Reference Platform hardware system. These capabilities are described in terms of "recommended" or "strongly recommended." Those capabilities necessary to run all operating systems are identified. Some capabilities, while recommended in some configurations for performance reasons, are only optional in other configurations because of size, cost, power consumption or
/ other considerations. In some cases, additional information is presented to help explain the implementation
/ of these requirements and recommendations. These requirements, recommendations, and miscellaneous
/ points are intended for construction of systems in the near term. As technology evolves they will be changed so that this specification stays current.

Hardware system vendors have to build systems that run effectively across the operating environments in which they expect to market their systems. It is possible to build a PowerPC Reference Platform-compliant hardware system that supports only one of the operating systems, but this is not a recommended approach.
/ By implementing all recommended system elements and features, hardware system vendors can ensure that
/ all target operating systems will be supported.

2.1 Processor Subsystem

/ This subsystem contains the processor(s) that operate on the data and instructions of the applications and
/ operating systems. Requirements, recommendations and miscellaneous information follow:

Requirements

- The processor subsystems for all compliant systems must comply fully with the PowerPC architecture.
| The PowerPC architecture is defined in *The PowerPC Architecture*, ISBN 1-55860-316-6. 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 processor subsystem time base (described in *The PowerPC Architecture*, Book III) must produce a minimum timing resolution of 500 nsec.
- The PowerPC 601 maintains a Real-Time Clock (RTC) rather than the time base. This Real-Time Clock must be driven by a 7.8125-MHz oscillator.

Recommendations

/ Processor bus frequencies of at least 20 MHz are recommended under normal operating conditions (e.g.
/ not in power savings modes).

Miscellaneous

- The architecture does not dictate the source of the time base frequency and other frequencies within the
/ system. The 603, for example, increments the time base once every four bus clocks of the local
/ PowerPC processor bus. In this case, the recommended minimum processor bus frequency of 20 MHz would support the time base resolution requirement.
/ • The PowerPC architecture requires that either a processor provide an implementation-specific interrupt
/ to software when the time base frequency is changed and provide a means to determine the current
/ update frequency, or the time base frequency must be under software control.

2.2 Memory Subsystems

Six memory subsystems are described in the following subsections: System Memory, System ROM, Non-volatile Memory, I/O Memory, System I/O, and External Cache.

2.2.1 System Memory

/ System Memory refers to the portion of the memory map for a system where executable instructions and
/ data for the applications and operating systems reside. Requirements, recommendations, and miscellaneous information follow:

Requirements

/ • A system must have a minimum of 8 MB of System Memory, but some operating systems may require
/ more.
- A system must provide for expansion of System Memory to at least 16 MB.
- System Memory must meet the coherency and serialization requirements defined in *The PowerPC Architecture*, Books II and III.
- The processors of a system must be able to read and write System Memory.
- The state of System Memory must be valid as long as power is applied to the memory subsystem.
- The System Memory must support the processor memory transactions for all target processors except as described in Section 3.17, “PowerPC Architecture Features Not Recommended.” All the transactions are defined in processor-specific user’s manuals.
/ • The memory controller for a system must fully decode the processor-generated addresses for System
/ Memory and must not have aliases.

Recommendations

- It is recommended that a minimum of at least 16 MB of System Memory be supplied on any system.
/ With this amount of memory, a hardware system will support any one of the operating systems that will run on PowerPC Reference Platform systems (refer to appendices).
- It is strongly recommended that a hardware system provide expansion capability of System Memory to at least 32 MB.
- It is recommended that System Memory be either parity checking or error checking and correcting (ECC).

Miscellaneous

- System Memory is normally attached to a memory controller which is located on the local primary processor bus. Expansions to System Memory are added directly to the same bus on which the base

System Memory exists. System Memory and expansions to System Memory may be located elsewhere as long as coherency is maintained. The most common implementation of main memory is DRAM.

/ • A system vendor may put I/O device memory in the System Memory area. This memory is not
/ reported as System Memory by the boot process, but is reported as part of the I/O adaptor information.
/ Operating systems will treat this memory as I/O Memory. Cache snooping is not required for addresses
/ in the range of the I/O device.

2.2.2 System ROM

/ System ROM contains the power-on and boot firmware and data required by the system. The size of System ROM is dictated by the size needed to hold the firmware required by the system. Typically System ROM is implemented using ROM, EPROM, or Flash ROM. Requirements, recommendations, and miscellaneous information follow:

Requirements

- A system must include a System ROM.
- The System ROM must be readable by the system processor.
- System ROM must maintain its state in the absence of system power.
- If System ROM is cached, then System ROM must support burst transfers to the target processor.

Recommendations

- It is strongly recommended that System ROM on all systems be writable by the system processor (e.g. Flash ROM).

Miscellaneous

- System ROM is not guaranteed to be accessible to the System I/O processors.
- 5.0, “Boot Process and Firmware,” describes the functions that the firmware in System ROM may perform.

2.2.3 Non-volatile Memory

Non-volatile Memory (NVRAM) is used to save system configuration and error indications across system boots. Requirements follow:

Requirements

- A system must contain a minimum of 4 KB of Non-volatile Memory.
- Non-volatile Memory must maintain its state in the absence of system power.
- Non-volatile Memory must be readable and writeable by the system processor.

2.2.4 I/O Memory

/ I/O Memory refers to the area of the memory map of a system where memory for devices resides. This memory is accessed by a system processor using load and store instructions. Examples of I/O Memory include graphics buffers, communications buffers, and I/O processor memory. Requirements and miscellaneous information follow:

Requirements

/ • Processor-generated addresses in the I/O Memory space must be converted by the system to the
/ addresses of the I/O device on the I/O bus.
/ • The system must convert processor loads and stores for I/O Memory addresses to transfers and com-
/ mands on the I/O bus.

Miscellaneous

- I/O Memory may exist on the system expansion bus(es) and is part of the I/O subsystems. When located on these buses it is typically not cached. I/O Memory may also be located on the primary processor bus. In this case, it will participate in the hardware-managed coherency protocol, unless other ports to the same area interfere.
- If I/O Memory were to be cached, then software would have to manage coherency. Some operating systems do not support software-maintained coherency. A design that required software to maintain I/O Memory coherency would exclude those operating systems.

2.2.5 System I/O

Part of the memory subsystem is configured to handle the addressing and communications for I/O devices. Within the PowerPC architecture, I/O can be performed by loads and stores to or from areas of the memory space which are mapped to the I/O addresses of devices. This area of the memory map is called “System I/O” and the technique of using it through loads and stores is called “memory-mapped I/O.” Requirements follow:

Requirements

- Processor-generated addresses in the System I/O memory space must be converted by the system to the addresses of the I/O device on the I/O bus.
- The system must convert processor loads and stores for system I/O addresses to transfers and commands on the I/O bus.

2.2.6 External Cache

An External Cache is a cache that resides between any on-chip processor caches and System Memory. Requirements and miscellaneous information follow:

Requirements

- If a system has an external cache, then that cache must follow the PowerPC architecture rules for maintaining coherency and serialization.
- If a system has an external cache, then that cache must be transparent to the software.

Miscellaneous

- An external cache may be included as an optional part of a system. Typically, these are level two (L2) caches, but may be level three (L3) or above if more than one level of cache is included on a processor chip.

2.3 Storage Subsystems

This section will describe four storage subsystems: hardfile, floppy, CD-ROM, and SCSI.

2.3.1 Hardfile

Requirements, recommendations, and miscellaneous information follow:

Requirements

- A system must have either a hardfile or hardfile capability (which is storage provided remotely via a network).

- If a system includes an internal hardfile, then the minimum size for that hardfile must be 120 MB of formatted uncompressed storage. Some operating systems and some configurations may require more storage space.

Recommendations

- It is strongly recommended that PowerPC Reference Platform systems capable of containing a hardfile have one with a capacity of greater than 200 MB of formatted uncompressed storage. This size will be sufficient to support any of the operating systems in their basic configurations.

Miscellaneous

- The minimum hardfile size was chosen to accommodate a client machine on a network with the smallest operating system currently being ported to this specification. Other operating systems and other configurations may require more.
- The hardfile capability may be achieved through direct connection such as SCSI or IDE, or it may be achieved through networking or an expansion adaptor.
- The hardfile capability may be achieved with rotating media or with other storage technology that are supported by the operating systems.

2.3.2 Floppy

Requirements, recommendations, and miscellaneous information follow:

Requirements

- If a system includes an internal floppy drive, then that drive must support 3.5-inch, 1.44-MB MFM-format floppies.
- Systems that provide floppy capability must achieve this capability through direct connection to a floppy drive and not through network attachment.
- If a system includes floppy drives and is shipped from the manufacturer after June 1, 1995, then the system must provide to software the capability to poll each drive up to one hundred times a second to determine the presence of media in the drive.

Recommendations

- It is strongly recommended that all systems capable of containing a floppy drive have one as standard. This capability enhances data sharing and delivery of loadable device drivers.
- It is strongly recommended that all floppy device drivers set both device select bits in the Drive Control Register to ones.

Miscellaneous

- An optional feature of floppy drives is auto-eject capability that allows the software to control ejection of the media.
- Currently most floppy device drivers leave both bits in the Drive Control Register set to 0 (e.g. Drive 1). In instances of polling with some floppy controllers, this setting may cause the access lamp to be lit.

2.3.3 CD-ROM

Requirements, recommendations, and miscellaneous information follow:

Requirements

- If a system includes a CD-ROM device, then system software must support at a minimum the ISO 9660 standard.

Recommendations

- It is strongly recommended that all systems capable of containing a CD-ROM have one as standard. This capability enhances software transport and application information content.
- It is recommended that CD-ROM drives be the double-speed drives capable of transfer speeds of at least 300 KB per second.
- If the CD-ROM is built into the system, it is recommended that the CD-ROM audio output be directly supplied to the input of the audio subsystem.

Miscellaneous

- For PowerPC Reference Platform-compliant systems that require CD-ROM capability, this may be achieved through direct connection such as SCSI or IDE. It may also be achieved through networking or expansion adaptor connection.

2.3.4 Storage System Interface

Recommendations follow:

Recommendations

- It is strongly recommended that the storage subsystem use the fast SCSI-2 interface to support hardfiles and CD-ROMs. This interface will also support external scanners, tapes, streaming tapes, optical storage, and RAID-based storage systems.

2.4 Human Interface Subsystems

This section addresses the human interface subsystems. These subsystems include the alphanumeric input device, the pointing device, audio and graphics.

2.4.1 Alphanumeric Input Device

Requirements, recommendations, and miscellaneous information follow:

Requirements

- A system must include an alphanumeric input device. By far the most common realization of this is a directly attached keyboard.
- A directly attached or built-in alphanumeric input device must be capable of generating at least 101 scan codes that can be interpreted by the machine-specific layer of the device driver.

Recommendations

- It is strongly recommended that the alphanumeric input device be a directly attached or built-in keyboard.

Miscellaneous

- No particular keyboard layout or interface is required.
- Most system environments require a directly attached or built-in keyboard. Those that do not require a directly attached or built-in keyboard are servers or multi-user systems with terminals attached.

2.4.2 Pointing Device

Requirements and miscellaneous information follow:

Requirements

/ • If a system has a directly attached or built-in keyboard, then it must also have a directly attached or
/ built-in pointing device.
/ • If a system includes a directly attached or built-in pointing device, then this device must provide two-
/ dimensional positioning as well as the capability of generating at least buttons up or down status.
• If the two-dimensional positioning information is absolute, then the device must report positioning information with at least the pixel resolution of the largest display supported by that system.

Miscellaneous

/ • Examples of pointing devices include mouse, Track Point II or III,* tablet, and touch screen.
/ • For devices that present relative two-dimensional positioning information, the systems software will
/ convert that information to positioning information with the pixel resolution of the display supported by
/ the system. The resolution of this device will affect its usability.

2.4.3 Audio

Requirements and recommendations follow:

Requirements

• A system must include audio capability.
/ • This audio capability must consist of at least two analog-to-digital input channels and at least two
/ digital-to-analog output channels.
/ • Each analog-to-digital and digital-to-analog channel must support sample widths of at least 16 bits.
/ • The audio capability must support at least sampling rates of 22.05 KHz and 44.1 KHz.

/ ***Recommendations***

/ • It is strongly recommended that at least one analog-to-digital input channel support a built-in monaural
/ microphone or jack for an external monaural microphone.
/ • It is strongly recommended that the digital-to-analog output channels support a jack for an external
/ stereo headphone.

2.4.4 Graphics

Requirements, recommendations, and miscellaneous information follow:

Requirements

/ • If a system has a directly attached or built-in graphics subsystem, then that subsystem must support at
/ least a 640x480, direct-mapped, 8 bits (e.g. 256 colors or gray shades) per pixel frame buffer.
/ • If a system is shipped from the manufacturer after June 1, 1995, then that system must support Bi-
/ Endian Graphics operations.
/ • If a system provides Bi-Endian graphics capability, then it must allow the system software to access the
/ graphics frame buffer in either Endian format.

Recommendations

/ • It is strongly recommended that the graphics subsystem support a color depth of 8, 16 or 24 bits and higher resolutions of at least 1024 by 768 pixels.

/ • If this recommended graphics resolution is not achieved by the display subsystem, it is recommended
/ that either the graphics adaptor or the graphics software support a virtual display capability that would
/ allow different parts of the frame buffer to be scrolled and panned smoothly through the display device.
• It is strongly recommended that the graphics subsystem support Bi-Endian operations before June 1, 1995.
/ • If a system includes Bi-Endian graphics support, then it is strongly recommended that the system soft-
/ ware be provided access to the registers for the graphics adaptor in either Endian format.

Miscellaneous

/ • Bi-Endian graphics support would allow Little-Endian graphics frame buffers to be switched to take Big-Endian multibyte pixels from Big-Endian operating systems (e.g. AIX*) without a processor-invoked byte reversal instruction. Bi-Endian graphics adaptor capability provides the broadest support for the target operating systems because at present Big-Endian operating environments require a Big-Endian view of graphics and Little-Endian operating environments require a Little-Endian view of graphics.
• The software interface to the graphics subsystem is accomplished through an implementation-bus-interface-specific device driver.
/ • Most PowerPC Reference Platform systems require directly attached or built-in graphics systems. An example of a system that does not require a directly attached graphics device is a server (e.g. data server, compute server, print server) that does not also have a user performing graphical operations and user interactions.
/ • VGA compatibility is not required, but some operating environments may need VGA support when
/ running some DOS applications.
• These graphics resolution requirements apply to the graphics adaptor and frame buffer. They should not be interpreted as graphics mode requirements or monitor/display resolution requirements.

2.5 Real-Time Clock

Requirements, recommendations, and miscellaneous information follow:

Requirements

• A system must include a Real-Time Clock (RTC) subsystem.
/ • The RTC must operate in the absence of primary power.
• The RTC must provide the necessary information to determine the year, month, day, hour, minutes, and seconds.
| • Operating systems that expect to participate in multiboot scenarios must maintain the RTC in
| Greenwich Mean Time (GMT).

Recommendations

/ • It is recommended that the accuracy of the RTC be at least +/- 0.001% of the seconds in a day. This accuracy is about +/- one second per day.

Miscellaneous

• The term “Real-Time Clock” (RTC) is also used in the descriptions of the PowerPC 601. The processor RTC does not replace and is different from this RTC.
| • In a multiboot scenario, several different operating systems would operate on a platform during different
| time spans. Maintaining the RTC in GMT provides a consistent known time base for each operating
| system.

2.6 Connectivity Subsystems

This section describes the serial, parallel and network connections for PowerPC Reference Platform-compliant systems.

2.6.1 Serial

Requirements, recommendations, and miscellaneous information follow:

Requirements

- A system must include at least one serial port.
- A system must implement this serial port using EIA/TIA-232-E signal compatibility.
- This serial port must support asynchronous protocol with baud rates up to at least 19.2 K.

Recommendations

- Support for baud rates higher than 19.2 K is recommended.

Miscellaneous

- An ASCII terminal connected to this serial port may provide alphanumeric input and output.

2.6.2 Parallel

Recommendations follow:

Recommendations

/ - It is strongly recommended that a system include at least one parallel port.

2.6.3 Network

Recommendations and miscellaneous information follow:

Recommendations

- If a system is to support low-end network communications, then LocalTalk (EIA-422-A) is recommended. LocalTalk is compatible with the SCC 8530 controller and is defined by interface standards and protocols.
- Where higher performance is required, the recommended LANs are either Ethernet or Token Ring*.
- It is strongly recommended that a system include an Ethernet connection.

Miscellaneous

- Other possible network connections include ATM, ISDN, FCS, FDDI, and Isochronous Ethernet.

2.7 Expansion Bus(es)

/ This section refers to the standard expansion buses that a hardware platform vendor may include in a
/ system. This specification does not require any particular expansion bus. Requirements, recommendations,
/ and miscellaneous information follow:

Requirements

- A system that contains one or more standard expansion buses must document the level of the bus standard to which it complies, as well as any deviations from that standard.

Recommendations

- It is recommended that vendors who implement an expansion bus use PCI, PCMCIA, and/or ISA. These buses are supported by the current operating system ports to PowerPC Reference Platform systems.

Miscellaneous

- Other buses that could be used with modifications to the abstraction software of each hosted operating system include VME, EISA, NuBus, VL, and MCA.

2.8 Additional Subsystems

There are other subsystems required in most computer system implementations. These subsystems include DMA, interrupt controller, timers, and system configuration registers. Requirements and miscellaneous information follow:

Requirements

- If a system contains a DMA controller, then that controller must not alias within the 32 bits of addressing.

Miscellaneous

- 6.0, “Reference Implementation,” provides one example of how these subsystems can be implemented from readily available components. Other implementations are possible. The ability to use these alternatives is limited by the ability of the abstraction software of the target operating systems to abstract the hardware.

2.9 Industry Interface Standards

This section lists standards that are applicable to the PowerPC Reference Platform subsystems. In many standards there are multiple incompatible implementations possible. This section addresses these issues so that implementations of these interfaces are consistent.

2.9.1 SCSI

Small Computer System Interface (SCSI) is an ANSI-standard specification for a peripheral bus. Requirements, recommendations, and miscellaneous information follow:

Requirements

- Systems that are implemented with SCSI must comply with the *ANSI Standard X3.131-1990 (Revision 10c) for SCSI-2*, (FAST SCSI). This standard specifies the electrical interface as well as the internal system connector.

Recommendations

- It is recommended that SCSI implementations use non-differential signalling with active termination.

Miscellaneous

- Use of this standard provides a convenient method for accessing CD-ROM, tape, hardfile, scanner and optical drives.

2.9.2 IDE

IDE is an optional interface for hardfiles. Requirements and recommendations follow:

Requirements

- Systems that are implemented with IDE must comply with the X3.221 AT Attachment (Revision 4.A); Proposed American National Standards.

Recommendations

- When the enhanced IDE interface becomes a standard and components are available, it is recommended that systems which use IDE use this interface for improved data rates and that system software use the capability for expanded capacity above 520 MB.

2.9.3 Ethernet

Requirements follow:

Requirements

- If Ethernet is implemented, it must adhere to the IEEE 802.3 standard. This specification covers both the electrical interface and the connectors.

2.9.4 Token Ring

Requirements follow:

Requirements

- If Token Ring is implemented, it must adhere to the IEEE 802.5 standard. This specification covers both the electrical interface and the connector.

2.9.5 Serial

Requirements, recommendations, and miscellaneous information follow:

Requirements

- The EIA/TIA-232-E standard for computer serial port connectors must be used for the required serial port.

Recommendations

- It is strongly recommended that compliant systems implement EIA/TIA-232-E using a nine-pin D-shell male connector and pin assignments as defined below.
- It is strongly recommended that systems use the pin configuration and signal assignments for the nine-pin serial port as shown in the following figure and table (viewed from the back of the machine).

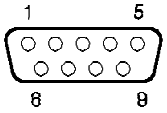


Figure 1. Nine-Pin Serial Connector Pin Arrangement

Table 2. Nine Serial Port Connector Signal and Pin Assignments	
Pin No.	**Signal Name**
1	Data Carrier Detect
2	Receive Data
3	Transmit Data
4	Data Terminal Ready
5	Signal Ground
6	Data Set Ready
7	Request To Send
8	Clear To Send
9	Ring Indicator

Miscellaneous

- The voltage levels are EIA only. A current loop interface is not supported.

2.9.6 LocalTalk

LocalTalk is the standard Macintosh serial port. Recommendations follow:

Recommendations

- It is strongly recommended that compliant systems implement EIA-422-A using the nine-pin connector and the following pinout.

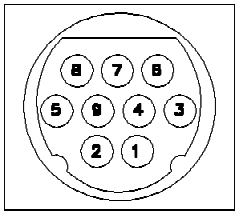


Figure 2. Connector -- Nine-Pin Arrangement

Table 3. Mini Nine-Pin Serial Port Connector Signal and Pin Assignments	
Pin No.	**Signal Name**
1	$SCLK_{out}$ Reset pod or get pod attention
2	$Sync_{in}/SCLK_{in}$ Serial clock from pod (up to 920 Kbit/sec)
3	TxD- Transmitted data inverted
4	Gnd/shield Signal Ground
5	RxD- Receive data inverted
6	TxD+ Transmitted data
7	Wakeup/TxHS Wakeup CPU or do DMA handshake
8	RxD+ Receive data
9	+5 V Power to pod (350 mA maximum)

2.9.7 Parallel Port Capability

Requirements and miscellaneous information follow:

Requirements

/ • If a system includes parallel ports, then those ports must be compliant to the standard specified by IEEE P1284, *Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers.*

/ ***Recommendations***

/ • It is strongly recommended that parallel ports of a system support the Extended Capabilities Port (ECP)
/ in addition to the Compatibility Mode.

Miscellaneous

/ • P1284 defines “compliant” as the “Compatibility Mode” and the “Nibble Mode.” The “Compatibility
/ Mode” is backward compatible with many existing devices, including the PC parallel port. The “Nibble
/ Mode” provides a bi-directional operations capability. The Extended Capabilities Port Mode (ECP) provides additional capabilities over this compatible mode.
• The Centronics interface is the popular name for the parallel printer port used by most “IBM compatible” personal computers. This interface has never been formalized.

2.9.8 PCI Bus

Miscellaneous information follows:

Miscellaneous

• PCI documents are available from and maintained by the PCI Special Interest Group. The PCI architecture is described in the following documents:
  – *PCI Local Bus Specification*, Revision 2.0, April 30, 1993
  – *PCI System Design Guide*, Revision 1.0, September 1993
/ – *PCI to PCI Bridge Architecture Specification*, Revision 1.0, April 5,1994
/ – *PCI Multimedia Design Guide*, Revision 1.0, March 29, 1994

2.9.9 PCMCIA Bus

Requirements and miscellaneous information follow:

The PCMCIA standard defines the physical requirements, electrical specifications and software architecture for the 68-pin cards and their sockets. Requirements, recommendations and miscellaneous information follow:

Requirements

- If a system has PCMCIA, then it must support the sockets that are release 2.x compatible.
- For maximum compatibility and interoperability, the system platform vendors must provide Socket Services and the operating system vendors must provide the Card Services extension.
- Both Socket Services and Card Services must be provided in the system abstraction software.
- The PC card vendors must provide Card Enablers (i.e. installers) which are clients of Card Services.
- If the operating system supports Plug and Play, then the Configuration Manager must take over individual Card Enablers.
- In addition to any default drivers, Card Services must allow for the use of Memory Technology Drivers to support additional types of memory devices.

Recommendations

- It is strongly recommended that the Card Services provide a default Memory Technology Driver.

Miscellaneous

- The PCMCIA software architecture has two key elements: Socket Services and Card Services. Socket Services is a hardware-dependent interface. The purpose of Socket Services is to mask the socket's actual hardware implementation from higher-level software components that utilize it. Card Services is a software layer that sits above Socket Services, coordinating access among the cards, the sockets and system resources such as interrupts and the memory map. Card Services accesses cards via Socket Services. The card drivers interact with the card via Card Services. In general, Card Services is operating system dependent.
- The PCMCIA standards can be ordered from and are maintained by the Personal Computer Memory Card International Association. The PCMCIA standards include the following individual specifications:
  - *PC Card Standard Specification*, release 2.1, July 1993
  - *Socket Services Specification*, release 2.1, July 1993
  - *Card Services Specification*, release 2.1, July 1993
  - *PC Card ATA Mass Storage Specification*, release 1.02, July 1993
  - *AIMS Specification*, release 1.01, November 1992
  - *Recommended Extensions*, release 1.00, November 1992
- In the future when the PCMCIA Card Bus Specification is approved, Card Bus should be considered if power consumption is within the allowable range.

2.9.10 ISA Bus

Requirements, recommendations, and miscellaneous information follow:

Requirements

- Systems must comply with the IEEE definition of ISA when it is approved.
- If Plug and Play is implemented on a system, then each Plug and Play hardware device must be able to be uniquely identified, must state the services it provides and the resources that it requires, must identify the driver that supports it, and must allow software to configure it.

Recommendations

- When the ISA Plug and Play standard is fully defined and accepted, it is strongly recommended that systems comply with the Plug and Play specifications. The Plug and Play specifications define hardware or software protocols to let systems perform configuration automatically.

Miscellaneous

- When the ISA bus was originally implemented, there was no architectural definition for the IBM Personal Computer AT Bus, only the implementation defined by the schematics published in the Technical Reference Manual. Since that time there have been several definitions of ISA put forward, including, but not limited to, the Family 1 Bus Architecture, an IEEE standards group, and an Intel publication.
- 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*.

2.9.11 Input Device Interfaces

Miscellaneous information follows:

Miscellaneous

- Examples of the interface for the alphanumeric and pointing devices include either ADB standard as used in Apple computers, or the PC/AT-PS/2* interface as implemented in an Intel 8042AH chip. The ADB interface circuit is described in the *Guide to Macintosh Family Hardware*.

2.10 System Configurations

Table 4 defines the required subsystem components for the minimum PowerPC Reference Platform-compliant system and recommended components for five typical computer configurations. Within this table, fields are marked as "None" to indicate that the device is not recommended, "Optional" to indicate that the hardware system vendor has the option to include the device, and "Required" to indicate that the device is required in all hardware configurations. The five configurations are:

Portable System	The portable configuration specifies the subsystems recommended for a PowerPC Reference Platform-compliant portable. A portable is a machine that is capable of battery operation.
Medialess System	The medialess system relies on network connections for all storage. Boot is from the network; software, data and paging space are attained from the network.
Desktop System	The desktop system is an entry-level system for commercial or technical applications.
Technical Workstation System	The technical workstation configuration specifies a technical user's desktop or deskside machine.
Server System	The server configuration specifies a machine that serves multiple users and does not require a locally attached keyboard and graphics display. Examples of functions performed by servers include backup server, compute server, data server, or print server.

Table 5 lists the minimum hardware configuration requirements for operating systems to run on a PowerPC Reference Platform hardware system. This table gives the requirements for the smallest version of the operating system required to support a stand-alone desktop workstation. Appendices for each operating system list the requirements for other configurations and list optional expansion and recommended features. The first column in Table 5 lists the minimum hardware configuration required to support any one of the oper-

ating systems. The remaining columns list the minimum hardware configuration requirements for each operating system.

Hardware system vendors must use the operating system information and the required minimum hardware configuration information in this section to configure a system. The hardware system vendor could use this information to tailor an implementation to include some operating systems in some configurations while excluding others, or could configure a machine large enough to hold any one or several of the operating systems.

Table 4 (Page 1 of 2). Subsystem Components of PowerPC Reference Platform Systems

Subsystem	Required Minimum	Recommended Typical Configurations				
		Portable	**Medialess**	**Desktop**	**Technical**	**Server**
Processor	PowerPC	PowerPC	PowerPC	PowerPC	PowerPC	PowerPC
System Memory	8 MB	Expandable to 32 MB (min.), parity	Expandable to 32 MB (min.), parity	Expandable to 32 MB (min.), parity	Expandable to 32 MB (min.), ECC	Expandable to 32 MB (min.), ECC
System ROM	Sized as Needed	Sized as Needed, Software Write-able	Sized as Needed, Software Write-able	Sized as Needed, Software Write-able	Sized as Needed, Software Write-able	Sized as Needed, Software Write-able
Non-volatile Memory	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB
L2 Cache	Optional	Optional	Optional	Optional	> = 256 KB	> = 256 KB
Hardfile	120 MB*	240 MB	None	240 MB	240 MB	400 MB
Floppy	Optional**	1.44 MB MFM	None	1.44 MB MFM	1.44 MB MFM	1.44 MB MFM
CD-ROM	Optional**	Optional	None	XA, >=300 KB/s	XA, >=300 KB/s	XA, >=300 KB/s
Alphanumeric Input Device	Required	Keyboard	Keyboard	Keyboard	Keyboard	Keyboard or Ter-minal
Pointing Device	Required with Keyboard	Required	2-Button Mouse	2-Button Mouse	2-Button Mouse	Required with Keyboard
Audio	16-Bit Stereo, 22.05 and 44.1 KHz	Multi-voice, Compression	Multi-voice, Compression	Multi-voice, Compression	Multi-voice, Compression	16-Bit Stereo, 22.05 and 44.1 KHz
Graphics	640x480x8	640x480x8	>=1024x768x16	>=1024x768x16	>=1024x768x16	Possibly Ter-minal
Real-Time Clock	Required	Required	Required	Required	Required	Required
Serial Port	EIA/TIA-232-E at 19.2K	> = 19.2K	> = 19.2K	> = 19.2K	> = 19.2K	> = 19.2K

Table 4 (Page 2 of 2). Subsystem Components of PowerPC Reference Platform Systems

Subsystem	Required Minimum	Recommended Typical Configurations				
		Portable	**Medialess**	**Desktop**	**Technical**	**Server**
Parallel Port	Optional	P1284 ECP Mode	P1284 ECP Mode	P1284 ECP Mode	P1284 ECP Mode	P1284 ECP Mode
Network	Optional**	Ethernet or LocalTalk	Ethernet or LocalTalk	Ethernet or LocalTalk	Ethernet or LocalTalk	Ethernet or LocalTalk
Expansion Bus(es)	Optional	PCI, ISA, or PCMCIA	PCI, ISA, or PCMCIA	PCI, ISA, or PCMCIA	PCI, ISA, or PCMCIA	PCI, ISA, or PCMCIA
SCSI	Optional	Optional	Optional	SCSI-2 Fast	SCSI-2 Fast	SCSI-2 Fast

Note:

***** Hardfile capability may be provided or augmented by a network connection, in which case a network connection is required.

****** Some method of installing the operating system, applications and data is required.

Table 5 (Page 1 of 2). Operating System Requirements for Subsystem Components

Subsystem	Required to Support Any One	Minimum Required for Each Operating System				
		Windows NT	AIX	Workplace	Solaris	Taligent
Processor	601, 603, 604	601, 603, 604	601, 603, 604	601, 603, 604	601, 603, 604	
System Memory	16 MB	16 MB	16 MB	8 MB	16 MB	
System ROM	Standard	Standard	Standard	Standard	Standard	
Non-volatile Memory	Standard 4 KB	Standard 4 KB	Standard 4 KB	Standard 4 KB	Standard 4 KB	
L2 Cache	Optional	Optional	Optional	Optional	Optional	
Hardfile	200 MB	200 MB	200 MB	120 MB	200 MB *	
Floppy (1.44 MB MFM)	Required	Optional	Optional	Required *	Required	
CD-ROM *	Required	Required	Required	Required	Required	
Alphanumeric Input Device	Required	Required	Required	Required	Required	
Pointing Device	Required	Required	Required	Required	Required	
Audio	Standard	Standard	Standard	Standard	Standard	
Graphics	800x600	640x480	800x600	640x480	640x480	
Real-Time Clock	Standard	Standard	Standard	Standard	Standard	
Serial Port	2 Required	Standard	2 Required	Standard	Standard	
Parallel Port	1 Required	Optional	1 Required	Optional	Optional	
Network	Optional	Optional	Optional	Optional	Optional	
Expansion Bus(es) **	ISA, PCI	ISA, PCI	ISA, PCI	ISA, PCI, PCMCIA	ISA, PCI, PCMCIA	

Table 5 (Page 2 of 2). Operating System Requirements for Subsystem Components

Subsystem	Required to Support Any One	Minimum Required for Each Operating System				
		Windows NT	AIX	Workplace	Solaris	Taligent
SCSI **	1 Required	Optional	1 Required	Optional	Optional	

Legend:

Entry	Definition
Standard	indicates that the component is required in all hardware configurations
Required	indicates that this component must be present to support that particular operating system
Optional	indicates that the operating system supports the component and that the hardware system vendor has the option to include the component
*	Capability can be provided via a network.
**	Currently supported devices. Different devices could be accommodated by providing alternative abstraction software.

3.0 Architecture Guidance

This section defines the collection of architectural constraints and conventions for PowerPC Reference Platform-compliant systems. Some features described within this section are required to support any operating system. These features are stated in terms of "must." Every PowerPC Reference Platform-compliant machine must have these features. Some features are recommended for better usability or performance. These features are described in terms of "recommended." Some possible features are discouraged for future
/ compatibility reasons and for hardware simplification reasons. In some cases, additional information is pre-
/ sented to help explain the implementation of these requirements and recommendations. This section is written primarily for hardware system vendors, but there are some software requirements and recommendations.

3.1 System Topology and Coherence

Miscellaneous information follows:

Miscellaneous

- A typical system topology used by PowerPC Reference Platform systems is shown in Figure 3. All PowerPC Reference Platform systems consist of one or more compliant processors; a volatile memory separate from other subsystems used for storage of data and instructions, called System Memory; and a number of objects, called I/O subsystems, that may initiate transactions to System Memory. The processors are linked over the *primary processor bus* to each other, to System Memory, and to a bus bridge. In general, I/O devices do not connect to the PowerPC processor bus. The bus bridge connects to a *secondary bus* which has I/O subsystems connected to it. In turn, another bus bridge may be employed to a *tertiary bus* with additional I/O subsystems connected to it. Typically the bus speeds and throughput decrease and the number of supportable loads increases as one progresses from the primary processor bus to the tertiary bus. PowerPC Reference Platform multiprocessor systems have a symmetric (at least from the hardware point of view) shared memory model. Please refer to Section 3.13, "Multiprocessor Considerations," for architecture considerations specific to multiprocessor systems.
- There are variations on this typical topology which are likely to occur and are therefore worth
/ describing. The secondary bus may be implemented as two or more parallel expansion buses for performance reasons. Similarly the tertiary bus may be two or more parallel expansion buses. The bus bridge and/or memory controller may be integrated into the processor chip. In the case where the bus bridge is integrated into the processor chip, the primary processor bus would not be the PowerPC processor bus, but would be an expansion bus (e.g. PCI). I/O subsystems would normally be attached to the primary processor bus in this case. If this bus is the only bus coming from the processor chip, then the System Memory would continue to be attached to the primary processor bus. If the memory controller was integrated into the processor chip, then System Memory would be attached to the processor chip.

3.2 System Memory

Requirements and miscellaneous information follow:

Requirements

- Access to System Memory by processors and I/O must conform to the coherency requirements specified by *The PowerPC Architecture*, when coherency is required.

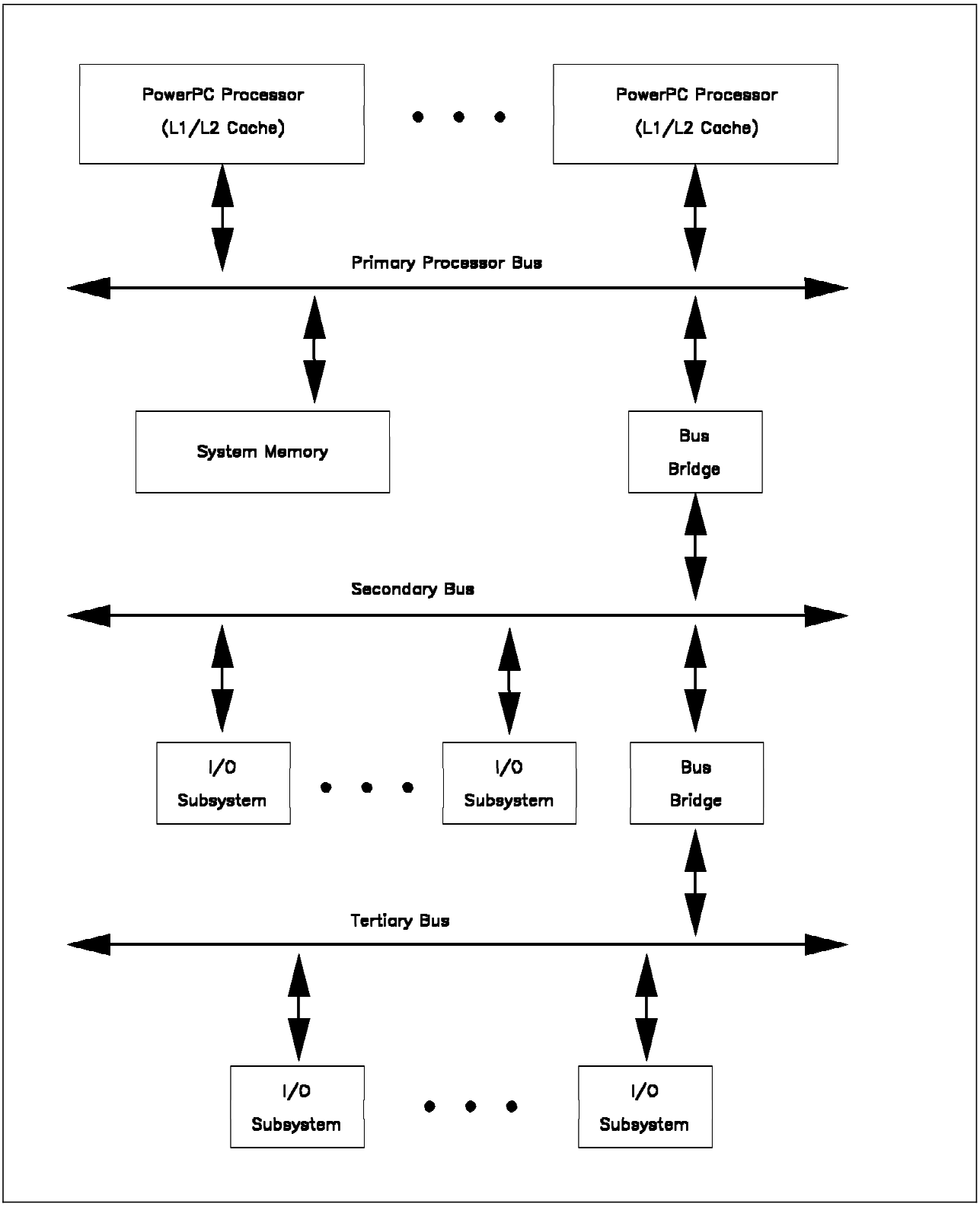


Figure 3. Typical PowerPC Reference Platform System Topology

- For coherence required pages, a coherent view of storage must be maintained between all processor caches and System Memory for any System Memory reference transactions initiated by the processor and for any transactions to System Memory initiated from an I/O bus. In particular:

- transactions between a processor and System Memory
- transactions between an I/O subsystem and System Memory
- transactions between any processor in a multiprocessor system and System Memory

Miscellaneous

- Coherence among any other storage elements or for any other kinds of transactions need not be maintained by hardware. In particular, transactions between an I/O device and I/O Memory need not be maintained coherent by the hardware.
- The PowerPC architecture allows software to set the coherence mode of System Memory as "memory coherence required" or "memory coherence not required." When memory coherence is required the hardware system maintains all data buffers in the L2 cache, in the memory controller, and in the bus bridge to the secondary bus in coherence.
- Some PowerPC processors may not assert the bus coherency tag bit (M bit) on instruction fetches; nevertheless, instruction fetches should be coherent with respect to prior write transactions between I/O devices and System Memory.
- Transactions between the I/O subsystem and System Memory are required to be coherent when the I/O operation is complete. An I/O operation is considered complete when the resulting I/O interruption processing is complete. In addition, most I/O bus architectures require strong ordering of I/O accesses to System Memory while an I/O operation is in process.
- Programmed I/O write transactions by a processor to an I/O device are required to be strongly ordered with respect to prior read transactions by that device to System Memory. Similarly, programmed I/O read transactions by a processor to an I/O device are required to be strongly ordered with respect to prior write transactions by that device to System Memory.
- Store combining (described in *The PowerPC Architecture*, Book III) by some PowerPC processors may have an effect on the order in which data is stored.

3.3 I/O Memory

PowerPC Reference Platform systems may contain CPU-visible read/write memory other than the System Memory. This memory is located on I/O devices which are normally attached to secondary or tertiary buses. These devices may be accessed through the bus bridge on the primary processor bus. Requirements and miscellaneous information follow:

Requirements

- I/O Memory which is not well behaved in terms of prefetching and other speculative storage operations must be marked as guarded.

Miscellaneous

- "Guarded" is a term used in *The PowerPC Architecture*, Book III. Storage which is not well behaved in terms of prefetching and other speculative storage operations must be marked as guarded. Examples of this type of storage include I/O subsystems, memory with holes, and the top of System Memory which has no successor. With a few exceptions, most of the I/O Memory would be mapped as guarded. Graphics frame buffers are an example of I/O Memory which would not be marked as guarded.
- The 601 treats all memory as not guarded in the sense that speculative fetching may occur to any area of memory. The 601 does speculative instruction execution. In this way, data which is adjacent to instructions could be speculatively executed. The 601 does not perform any speculative load or store instructions.

3.4 System I/O

Requirements, recommendations, and miscellaneous information follow:

Requirements

- Software must assure that processor implementations of load and store combining do not violate bus and device operational constraints.
- All transfers initiated by a processor, independent of the target of the transfer, must obey PowerPC architectural semantics for memory ordering (including semantics for *eieio* and *sync*).

Recommendations

- It is recommended that all loads or stores to System I/O match the device register size.
- If load and store combining by the processor would violate bus and device operational constraints, then it is recommended that software separate adjacent loads and stores with storage synchronizing instructions (e.g. *eieio*).

Miscellaneous

- The PowerPC architecture allows processors to implement “load or store combining.” This architectural feature allows a processor to combine one or more loads or stores to adjacent storage into a single bus transaction. In this way, two store word instructions to adjacent locations could become a doubleword bus transfer.

3.5 Self-Modifying Code

Programs which modify the currently executing sequence of instructions, self-modifying code, are allowed on PowerPC Reference Platform systems. Requirements, recommendations, and miscellaneous information follow:

Requirements

- Self-modifying code must be done in such a manner that the instruction fetch pipeline and cache do not contain the original form of the modified code.

Recommendations

- It is recommended that operating system software provide a service which will allow applications that modify code to ensure that instructions and data in cache and System Memory are coherent.
- The recommended instructions for obtaining instruction and data coherence are as follows:
  a) *dcbst* -- update System Memory
  b) *sync* -- wait for update
  c) *icbi* -- remove (invalidate) copy in instruction cache
  d) *sync* -- wait for *icbi* to be globally performed (may be omitted in uniprocessor systems)
  e) *isync* -- remove copy in own instruction buffer
- Because of the overhead incurred while performing the instruction and data storage synchronization, it is recommended that frequent minor self-modification to the instruction stream be avoided.

Miscellaneous

- The PowerPC architecture does not enforce consistency between the instruction cache and System Memory.

3.6 Resource Locking

The *PowerPC Reference Platform Specification* does not require resource locking capability beyond the atomic access features provided in the PowerPC architecture. Some buses provide a resource locking function (e.g. PCI exclusive access mechanism) for use by bus masters. System designs may choose not to implement resource locking functions. If resource locking is provided, then the following requirements, recommendations, and miscellaneous information apply:

Requirements

- If resource locking is provided for bus masters, then systems must not allow the entire bus to be locked.
- Locking of a system memory resource by bus masters is acceptable, but memory system coherency must always be maintained.
- Bus bridges which support the locking of system memory between devices and device drivers or other processor tasks must present a read-with-intent-to-modify transaction to the processor interface when a device attempts to lock a system memory resource.
- Locking of system memory resources by bus masters must occur in increments of less than or equal to a memory system page (4 KB), and must not cross a page boundary.
- Processor operations which require the use of locking features must use the PowerPC atomic access feature (e.g. *lwarx* and *stwcx*).

Recommendations

- Locking of system memory in cache line increments and aligned on cache line boundaries is recommended.
- If resource locking is implemented for a given bus architecture, then it is recommended that the implementation fully support the resource locking specification for that architecture.
- Other than the atomic access features provided by PowerPC processors, additional hardware support for locking operations may vary among systems. It is recommended that devices and device drivers be designed to work in the absence of such support.

Miscellaneous

- Requiring a read-with-intent-to-modify transaction to be presented to the processor interface when locking a system memory resource between a device and its device driver causes cached elements affected by the lock to be flushed and invalidated before the lock is granted. It will also cause reservations to a cached element affected by the lock (e.g. those obtained by a processor via a *lwarx* instruction) to be cancelled. Subsequent attempts to access the resource, except by the lock owner, can then be retried while the lock is valid.
- Lock operations will generally be of three types:

  a) locking of a slave device by a master

  b) locking a system memory resource between two masters

  c) locking a system memory resource between a master and its device driver

  It is likely that the third type will be the most widely used by devices.

3.7 Bus Errors and Unsupported Bus Transactions

Requirements and miscellaneous information follow:

Requirements

- If bus errors (e.g. detected parity errors) and unsupported transaction types (e.g. I/O of an unaligned 4-byte word) to local and expansion buses are to be reported, then the system must send a high-priority interrupt (e.g. soft reset, NMI, INT0, or Transaction Error Acknowledgement -- TEA) to the PowerPC processor.

Miscellaneous

- The level of error reporting in PowerPC Reference Platform-compliant systems is not defined in this specification, but is left up to the system designer. A minimal-cost implementation may choose to ignore error conditions and continue processing. A more robust design may detect and report error occurrences to software.
- When a PowerPC processor receives a TEA, it produces a "machine check" interrupt. The PowerPC architecture allows the implementation to define whether interrupts are "precise" or "imprecise." The PowerPC 601, 603, and 604 have defined the soft reset and machine check interrupts as imprecise, which means that the exact instruction address that caused the interrupt may not be reported to the interrupt handling software. Recovery from imprecise interrupts may be difficult or impossible. In most cases the interrupt handler may have to record the error conditions in NVRAM and stop the system.

3.8 Memory Map

Requirements, recommendations, and miscellaneous information follow:

Requirements

- The memory map must consist of four distinct areas, including:

System Memory	Used to store the programs and data being operated on by the processor
System I/O space	Used for input and output to devices attached to the I/O buses
I/O Memory	Memory on I/O devices such as local memory on graphics adaptors
System ROM	Contains the initial bring-up code

- The size and location of these memory spaces must be passed to the operating system loader in the residual data.
- System Memory (as real memory) space must start at 0.
- If the IP bit of the MSR is 0, then the area of System Memory from 0 through X'00002FFF' must be reserved for the interrupt vector table as defined by the PowerPC architecture.
- If the IP bit of the MSR is set to 1, then the area of the memory space from X'FFF0 0000' to X'FFF0 2FFF' (X'FFFF FFFF FFF0 0000' to X'FFFF FFFF FFF0 2FFF' in 64-bit processors) must be reserved for the interrupt vector table as defined by the PowerPC architecture.
- System ROM must be located at the top of the memory space because the processors begin execution after power is turned on at X'FFF00100' (X'FFFFFFFFFFF00100' in 64-bit processors).

Recommendations

- The following describes the recommended approach that the boot process would use to pass the memory map to the operating system. The firmware and system registers implemented by the supporting hardware would contain enough information about the memory map to allow the bring-up process to construct the memory map. The bring-up process would extract this information, discover any holes in memory, and describe the memory map in the residual data. Section 5.6.1, "Map of Residual Data Structure," shows the residual data map which contains the structure "MEM_MAP" that is used to pass the memory map to the operating system. This abstraction of the memory map will allow variations from implementation to implementation and will provide for expansion of addressing to 64 bits.
- It is strongly recommended that memory map implementations accommodate the dedicated real memory required by AIX as described in Appendix E.7, "Boot Time Abstraction Requirements."

Miscellaneous

- Two example address maps are described below to demonstrate possible arrangements. These particular arrangements are not required by the *PowerPC Reference Platform Specification*.

3.8.1 Example Memory Maps

Figure 4 shows one variation of a PowerPC Reference Platform System Memory map. The left side of this figure shows the view of memory from the PowerPC processor. The right side of this figure shows the view / of memory of the I/O master doing I/O addressing or memory addressing. The large arrows in the figure / show correspondence of areas and not data flow which, in some cases, may be in only one direction. As shown on the left side of the figure, the address space is split into four areas: a System Memory portion with addresses from 0 to 2 GB, a System I/O portion which stretches from 2 GB to 3 GB, an I/O Memory area which covers 3 GB to 4 GB - 16 MB, and a System ROM and Register space from 4 GB - 16 MB through 4 GB.

The System Memory is addressed by the operating system and user applications executing in the PowerPC processor. Access to System Memory by the PowerPC processor is in the 0 to 2 GB range. System Memory accessed by an I/O master will be addressed in the 2 to 4 GB range as shown on the right side of the figure.

The System I/O address space is used for input and output to the I/O subsystems attached to the secondary or tertiary I/O buses (e.g. PCI, VL, ISA, or Micro Channel).

The I/O Memory area is used for graphics memory and other I/O-based memory.

The System ROM and space for system registers is allocated to the top 16 MB of the memory map. The System ROM contains the code for power-on self tests (POST), code for establishing the initial configuration of the system, code for bring-up, vital product data (VPD), and system-specific information.

As shown on the right side of the figure, the I/O master has two modes of addressing. The I/O master doing memory address mode transfers will see System Memory as having addresses from 2 GB to 4 GB. I/O masters addressing I/O Memory on adaptors such as VRAM will address this I/O Memory in the range of 0 to 1 GB. The upper 16 MB of this space is reserved for the System ROM and system registers and is not addressable by the I/O master. In the memory-addressing mode, the I/O master cannot address the space from 1 GB to 2 GB. The I/O master doing I/O addressing mode transfers will see the System I/O space as having addresses from 0 to 1 GB.

Within this architecture, treatment of addresses outside of the allowed ranges is implementation dependent.

Figure 5 shows a different PowerPC Reference Platform memory map. The first 640 KB contain intra-system communication information such as the stack, scratch pad, interrupt vectors, residual data from the bring-up process, and interprocess messaging areas. The space between 640 KB and 1 GB is used for I/O Memory such as graphics buffers, cache for storage devices, network buffers and video buffers. System Memory runs from 1 GB to 3 GB. Memory space for System I/O runs from 3 GB to 4 GB - 16 MB. This area is used for memory-mapped I/O to devices on the I/O buses. System ROM and system registers occupy the space from 4 GB - 16 MB to 4 GB.

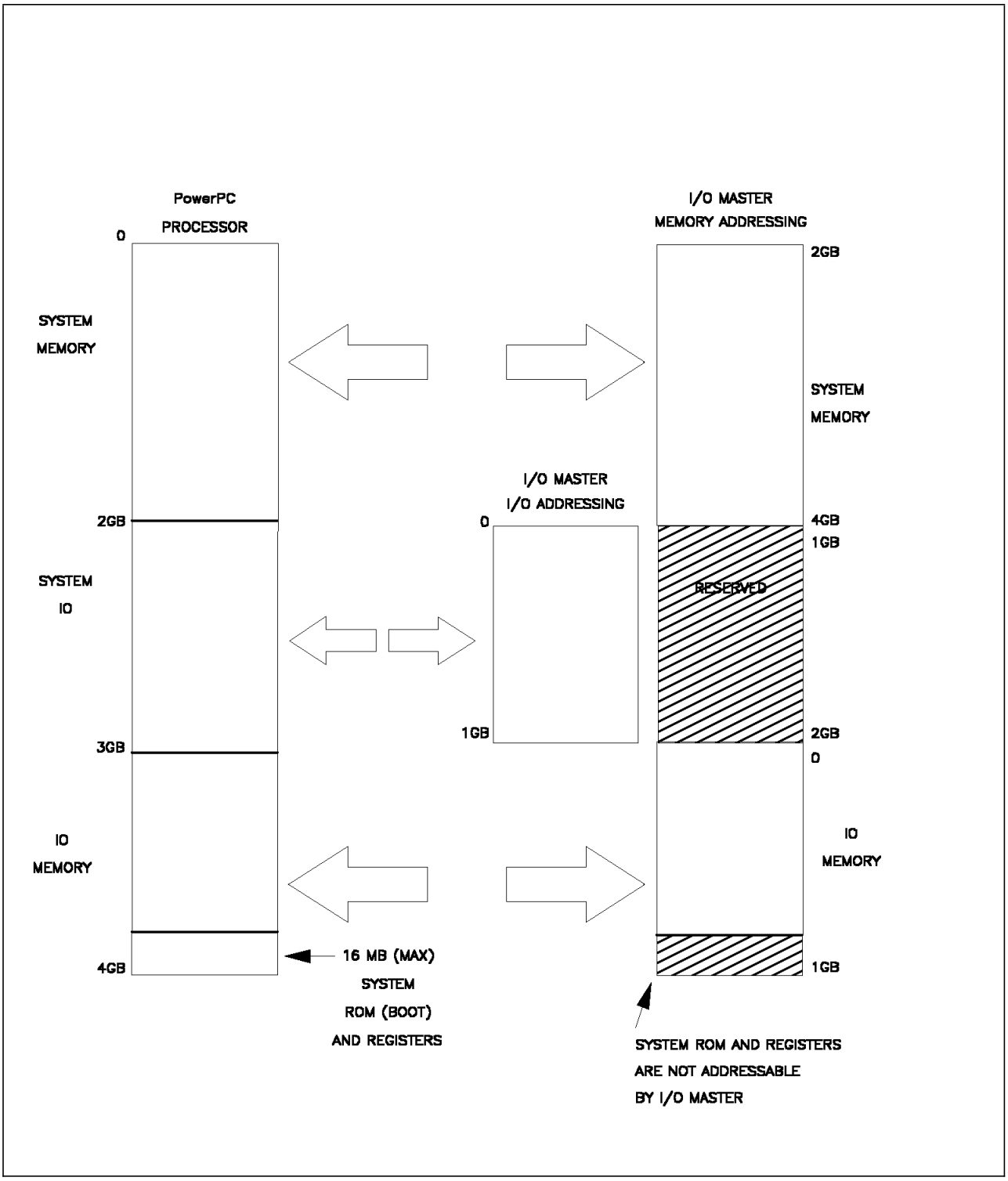


Figure 4. Example of a Memory Map -- Alternative 1

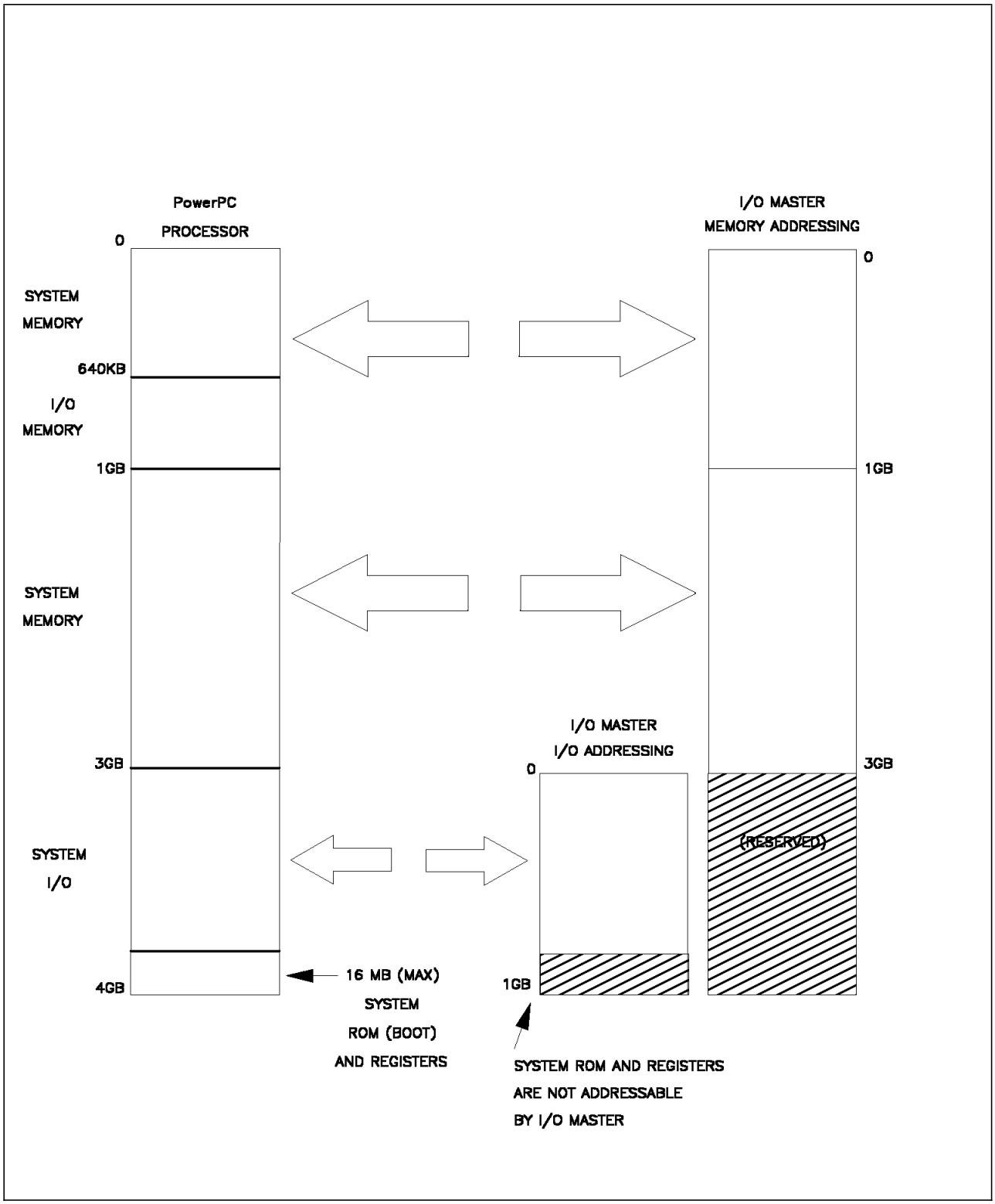


Figure 5. Example of a Memory Map -- Alternative 2

3.9 Memory Ordering

Requirements and miscellaneous information follow:

Requirements

- Since some processors support only a weakly ordered memory model, software must program for that case.
- All transfers initiated by a processor, independent of the target of the transfer, must obey PowerPC architectural semantics for memory ordering (including semantics for *eieio* and *sync*).

Miscellaneous

- A machine conforms to a weakly ordered memory model if:
  - accesses to global synchronizing variables are sequentially consistent, and
  - no access to a synchronizing variable is issued in a processor before all previous global data accesses have been performed, and
  - no access to global data is issued by a processor before a previous access to a synchronizing variable has been performed
- "Sequentially consistent" as used above means that a load or store for any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program. A store is completed when the value stored by the processor executing the instruction can be seen by all other processors. A load is completed when the value to be returned by the load has been set and cannot be changed.
- Thus for a weakly ordered memory model, the order in which a processor performs storage accesses, the order in which those accesses complete in main storage, and the order in which those accesses are seen as completing by another processor may all be different.
- Storage accesses by a single processor without explicit synchronization operations may not appear to complete in program order when viewed from another processor. However, accesses to the same address from the same processor will complete in program order.
- The instructions *eieio* and *sync* may be used to enforce ordered storage access with respect to a given processor. Higher-level protocols (using *lwarx* and *stwcx* for example) must be used to ensure ordering between processors or between processors and devices.
- The operations *lwarx* and *stwcx* are defined only on effective addresses that are mapped to System Memory on the local PowerPC processor bus. (Note that PowerPC architecture does not specify that *lwarx* or *stwcx* necessarily generate externally visible transaction. Consequently, these restrictions on *lwarx* and *stwcx* are not enforceable in hardware.)
- For additional reading on memory ordering, refer to *Memory Access Buffering in Multiprocessors* by Michel Dubois, Christoph Scheurich, and Faye Briggs, in 13th ISCA, pages 434-441, 1986.

3.10 Configuration and Diagnostics

Requirements, recommendations, and miscellaneous information follow:

Requirements

- A PowerPC Reference Platform system must include some form of power-on self test.
- In addition, stand-alone diagnostics must be supplied for systems.

Recommendations

- It is recommended that stand-alone diagnostics be operating system independent.
- It is strongly recommended that each operating system support on-line configuration and concurrent maintenance (on-line diagnostics).

Miscellaneous

- While power-on self test and stand-alone diagnostics are sufficient for some customers' needs, many others may require on-line configuration and diagnostics. On-line configuration and diagnostics are indicated by some of the likely attributes of near-term computer systems including:
  - increased power management
  - hot-plug capability
  - continuous availability
- A highly power-managed system is not often shut off or restarted. Thus the initial hardware checkout, test, and verification (e.g. power-on self test) and stand-alone diagnostics cannot realistically be relied upon to provide the primary means for system maintenance and diagnostics. Properly, this task should be part of the run-time environment supplied by an operating system.
- Hot-plug allows a hardware subsystem to be dynamically coupled or decoupled from the system. This will require the run-time environment supplied by the operating system to configure and verify the operation of any newly coupled device. It is likely that this function would be included in the respective device drivers. This function also complements the on-line diagnostics function. The result of diagnosis might be removal, replacement, and configuration of some I/O subsystem while the system is on-line.
- Continuously available systems are becoming more important. Workstations can fill the role of servers which could run continuously unattended. Workstations running as process monitors (e.g. ATM or process control) are other examples. In this environment, on-line maintenance and on-line configuration are both important attributes of the operating environment.
- Refer to Section A.7, "A Proposed Diagnostic Strategy," for a description of an implementation of diagnostics support.

3.11 Power Management

Two types of power management techniques have been used in the computer industry. The first type is hardware managed and is inaccessible to system software. This approach will be referred to as "micro power management." The second type uses system software to control hardware and will be referred to as "macro power management." Micro power management is the predominant technique in the existing base of PCs. This is due largely to the need for power management in portable computers before operating systems offered power management support. Macro power management is by far the more powerful technique and is thus the basis for the Reference Platform power management model.

Requirements, recommendations, and miscellaneous information follow:

Requirements

- If a hardware platform supports power management, then its power management features must be controllable by system software.
- Abstraction software and device drivers must provide the necessary functions to support the power management features of its target operating system.

Recommendations

- It is strongly recommended that systems incorporate power management.
- It is recommended that system software and hardware work effectively together to provide the necessary functions to implement a macro power management model similar to that shown in Figure 6. In this model, the following is true:
  - The operating system is the power management control hub.
  - Power management policy is set by the user through an application interface.
  - The operating system implements power management policy through the system abstraction and device driver interfaces.
  - The operating system supports power-management-aware applications.

- It is recommended that operating systems abstract the following activities, and software abstraction services and device drivers support these activities:
  - Issuing power state change commands to the system, subsystems, and devices.
  - Monitoring devices and subsystems for power management event notifications and providing approval.
  - Receiving power management requests from devices and subsystems.

Miscellaneous

- Abstraction software and device drivers provide the connection between the operating system and hardware. This software is aware of hardware power management capabilities and translates operating system power management requests into specific device and subsystem controls. It also passes information provided by hardware to the operating system to help it perform power management.

3.11.1 Support for Suspend and Hibernation System States

This specification does not mandate particular system power states or algorithms for transition between states. Two states, however, will likely be implemented in most systems. These states require special consideration when designing hardware, abstraction software, device drivers, and operating systems. The first is a suspend state which is characterized by very low power dissipation and fast recovery to full function. Resumption from suspend generally occurs upon receiving an I/O interrupt resulting from mouse or keyboard activity, communications events, or when opening the lid of a laptop system. In most systems the contents of system memory will remain active while in the suspend state. Hibernation is the second state, and is defined as the system being turned off, with the system state being written to non-volatile storage such as hardfile. Recovery from hibernation would generally not be as quick as that from suspend, but it saves the time of loading the operating system and brings the system back to its state before it hibernated. Resumption from hibernation would occur when the system is powered back on.

Requirements, recommendations, and miscellaneous information follow:

Requirements

- Power-managed systems must provide the following infrastructure to support a suspend state:
  - Hardware must provide a software-writeable suspend flag. This bit can be held in NVRAM or in a register which maintains its value while the system is in the suspend state.
  - Hardware must provide a software-writeable resume pointer to indicate to firmware the correct starting point when resuming from suspend.
  - If there is a firmware requirement for a free contiguous memory space when resuming from suspend, then this must be communicated to system software.
- Compliant operating systems must abstract the location of the suspend flag and the resume pointer.
- Prior to suspending, the operating system must allocate the firmware requirement for free contiguous memory.
- If the hibernation state is implemented, the OS must set a hibernation flag prior to powering the system off to indicate to the OS loader that a saved system state exists.
- After resuming from hibernation or suspend, the operating system is responsible for updating its time of day using the system's persistent time-keeping mechanism.

Recommendations

- It is recommended that operating systems implement the system suspend and hibernate functions.
- It is recommended that firmware verify that the saved memory image is valid before jumping to the resume address when resuming from suspend.
- It is recommended that the saved system state is verified before loading it when the hibernation flag is set.

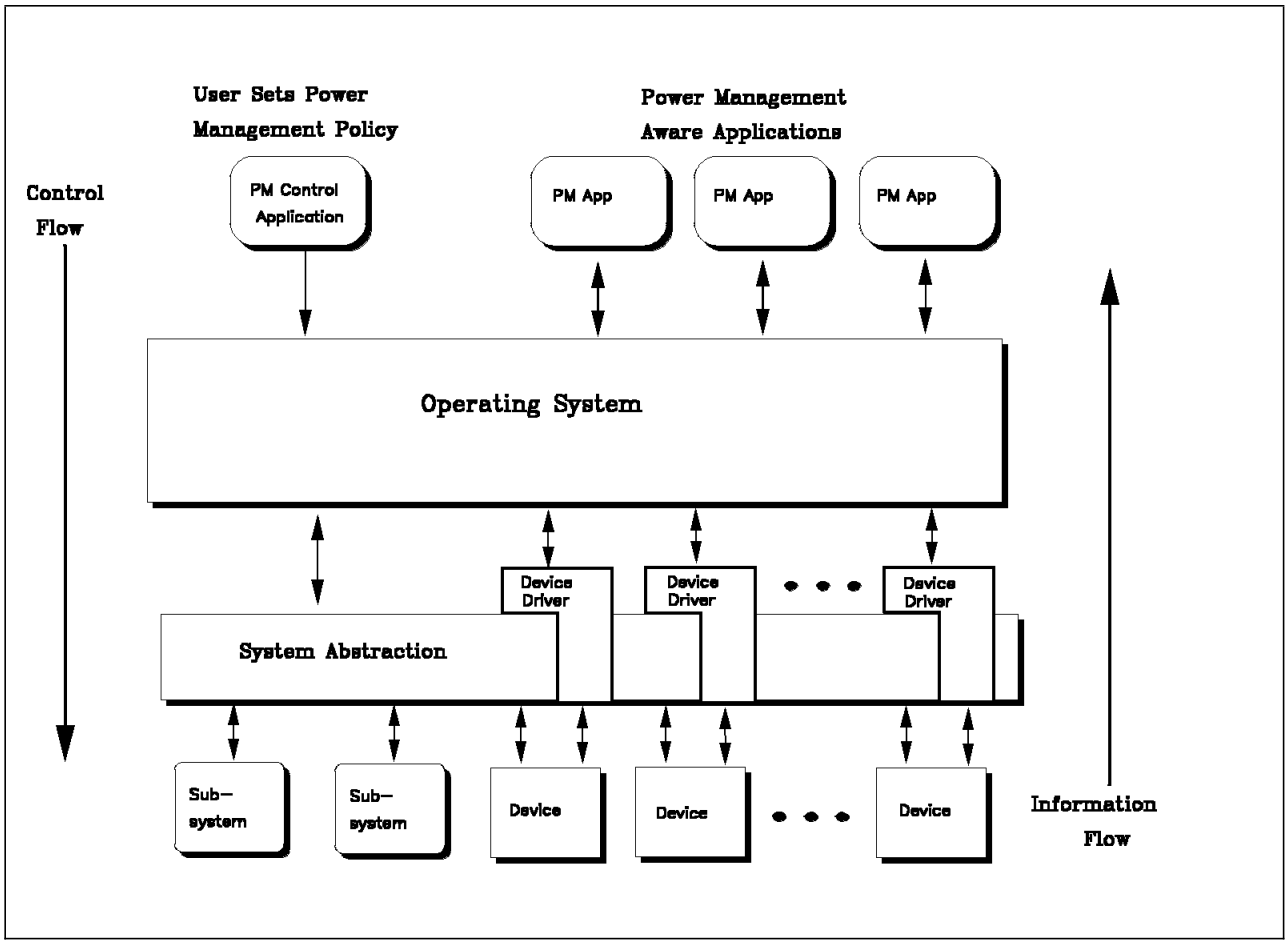


Figure 6. Macro Power Management Model

- It is recommended that power-managed systems provide a software-controllable main power switch which would allow automatic hibernation when the system is powered off or when the battery supply is critically low in portable systems.
- It is recommended that the hibernation flag be kept on the media that holds the saved system state. This approach may avoid confusion if that media was either removed from the system or was not operational after the system had been hibernated.

Miscellaneous

- To accommodate suspend and hibernation requirements, the NVRAM provides the following fields:
  - A 1-byte field which could be used to hold the suspend flag (called _PM_MODE, see Section 5.5.5, "Map of NVRAM Data Structure"). If this field is used for the suspend flag, then verifying the contents of memory is very important. This is due to the possibility of system power being lost during the suspend time frame. If the system is rebooted, the flag in NVRAM would remain set. Another alternative would be to use a special purpose register to hold this flag, which would default to the reset state if system power was lost and then restored.
  - A 32-bit resume pointer field (a member of the struct _RESTART_BLOCK called RestartAddress, see Section 5.5.5, "Map of NVRAM Data Structure"). This field is used to tell firmware the correct starting point to jump to when passing control to system software.
  - A 32-bit pointer (also a member of the _RESTART_BLOCK called SaveAreaAddr, see Section 5.5.5, "Map of NVRAM Data Structure"). Some firmware implementations may require a fixed amount of free contiguous memory to resume from suspend. The operating system would be required to free this space before setting the suspend flag. This OS would use this pointer to tell firmware the start address of this space.

3.11.2 Device and Subsystem Power Management

Effective system power management depends on subsystems and devices which provide power management support. This usually means support for multiple power states and hooks for system software to control those states. Some devices and subsystems may implement micro power management techniques, in addition to any macro power management capabilities they provide. A good example of this is the PowerPC 603 processor's management of the floating-point unit. It will turn the floating-point unit off and on based on the instruction stream it is executing.

Requirements, recommendations, and miscellaneous information follow:

Requirements

- If a device or subsystem is micro power managed, then these techniques must not affect the correct operation of programs.
- When entering the suspend state, the CPU core must be the last device to power off and the first device to power on when resuming. The CPU core is the hardware necessary to execute instructions, and includes the processor, clock, cache, and primary processor bus.

Recommendations

- The following device and subsystem modes are recommended as a minimal set for macro power management. The definitions are intended as a general guideline and will vary depending on the device or subsystem.
  - *On*:
    - The device or subsystem is fully powered and able to perform at maximum performance.
  - *Power Managed*:
    - The device or subsystem is working, but some or all features may be functioning at reduced performance levels.
    - Power is maintained.
    - State information is retained.
  - *Low Power*:
    - The device or subsystem is not operational, but can resume to the power-managed state quickly.
    - Some power is maintained.
    - Device state information is retained.
  - *Off*:
    - The device or subsystem is not operational.
    - The device or subsystem is powered off.
    - State information is not retained.

Miscellaneous

- The use of micro-power-managed devices requires careful consideration to avoid incorrect program operation. Two areas of particular concern are making sure system buffers are written to disk before powering off logic which contains them, and device or subsystem response time while running time-critical applications.

3.11.3 Power Management Hardware and Abstraction Software Features

Requirements, recommendations, and miscellaneous information follow:

Requirements

- The processor timekeeping (e.g. Real-Time Clock in 601 and time base in other PowerPC processors) must be accounted for in the hardware and software systems. This implies the following:

/ – If the processor timekeeping functions are going to be slowed down as in the power-managed state,
/ then this frequency change must be accounted for when calculating time of day or elapsed time to an
/ interrupt.
/ – If the processor timekeeping functions are going to be stopped as in the power-managed -- off state,
/ then time-based interrupts (e.g. decrementer) and the time of day which must persist across the
/ power off period must be maintained in other hardware within the system.
/ • Each device must provide for its state to be restored.
/ • If a device's state registers are implemented as write only, then the device driver must maintain a shadow
/ copy of these registers.

Recommendations

/ • It is recommended that only systems implementing Open Firmware use the technique of slowing the
/ processor clock to manage power consumption. Switching the clock between its normal frequency and
/ off is acceptable.
• It is recommended that device registers be read/write for use with save and restore functions.
• It is recommended that components of the system provide idle and/or usage indications to the power management controller software.
/ • It is recommended that the system preserve security features during power management. For instance, if
/ the system has a mechanism
/ to check for a user password when powering on from an "off" state, then this facility should also exist
/ when powering on from a "hibernate" state.
• It is recommended that systems using DRAM be designed to allow the refresh rate to be slowed down or switched to a self-refresh mode. In the self-refresh mode, the memory controller may be shut down.
• For devices with counters, it is recommended that the internal counter's clock be gated off and not used except when required for the counter.
• It is recommended that memory controller designs have the following attributes:
  – All registers should be read/write.
  – Support CAS before RAS refresh in order to allow low-power DRAM refresh. This mode may be used in both normal and suspend mode.
  – Use RTC clock at 32 KHz as the refresh clock in suspend mode.
  – Tri-state all address, data and control signals except the memory interface in suspend mode.
  – Stop clock to all logic except memory interface in suspend mode.
• If a PCI bus is present, it is recommended that the PCI clock, which is programmable, be turned off in suspend mode.

/ ***Miscellaneous***

/ • Open Firmware will provide methods to query devices and subsystems about their power management
/ capabilities, including the frequency at which the processor will run in its different power-managed states.
/ An architected means to communicate this information to the operating system prior to implementing
/ Open Firmware does not exist.

3.12 Bi-Endian Support

Requirements and miscellaneous information follow:

Requirements

• A system must support two modes of operation: Big-Endian and Little-Endian.
• The hardware system design must provide the capabilities for a Big-Endian operating system (e.g. AIX) and its applications to run efficiently or for a Little-Endian operating system (e.g. Windows NT** or WP/OS) and its applications to run efficiently.
/ • A system must provide hardware which presents the data to the processor in the proper format in both
/ Endian modes.

- A system must include hardware which presents the data and addresses to memory and I/O in the proper format in both Endian modes.

Miscellaneous

- A system that runs operating systems and applications software and uses data that are stored either Big-Endian (e.g. big end first -- most significant byte first) or Little-Endian (e.g. little end first -- least significant byte first) is referred to as "Bi-Endian."
- This Bi-Endian requirement does not mean that simultaneous operation of Big-Endian and Little-Endian, called "Mixed-Endian," is required. For instance, a Little-Endian operating system running a Big-Endian application without the aid of emulation and translation software is a Mixed-Endian environment. Eventually this Mixed-Endian capability may become practical, but currently the synchronization at mode switching time degrades performance.
- Because the current PowerPC processors assume that multibyte scalars are placed in storage in Big-Endian byte order, there must be some translation somewhere in the system when the processor references objects (programs, data, etc.) which are Little-Endian. Refer to Appendix B, "Bi-Endian Design Guidance," for a description of Bi-Endian design alternatives. Designs which are applicable to the current set of processors (e.g. PowerPC 601, 603, 604, and 620) would have the following attributes:
  - The designs would provide hardware which reverses the location of each byte in each doubleword when in Little-Endian mode.
  - The designs would provide hardware which modifies (e.g. returns the address to its original value) the address of a scalar generated by the processor when in Little-Endian mode.
  - The designs would provide these hardware functions together either on the path between memory and I/O to the processor or on the path between the I/O devices and memory and the processor.

3.12.1 Software Support for Bi-Endian Operation

Requirements, recommendations, and miscellaneous information follow:

Requirements

- Each Little-Endian operating system must perform an Endian mode switch during the boot process.
- The Endian mode switching instructions must be provided by the abstraction software for each Little-Endian operating system.
- Software support for the processor capabilities to perform byte reversal operations must be provided.

Recommendations

- It is recommended that software support both the Bi-Endian Memory and I/O design (refer to Figure 56) and the Bi-Endian I/O design (refer to Figure 57).
- It is recommended that device drivers and other service code be written in an Endian-neutral manner.

Miscellaneous

- At power on, or after a system reset, the processor is in Big-Endian mode. The particular instructions to perform an Endian mode switch are processor and hardware system dependent.
- In some cases software may have to byte-reverse data before sending it to an I/O device. For instance, Big Endian device drivers sending multibyte control data to Little-Endian registers on an I/O device must byte-reverse this data in the processor.
- Load and store with byte reversal of 2- and 4-byte scalars are contained in the PowerPC architecture. Invoking these instructions should be supported by language syntax and compiler support. If the compilers for all languages do not support these forms of loads and stores, then the runtime library routines should supply services which perform the byte reversal.
- "Endian-neutral." is defined as software which is aware of the Endianess of the system and is designed for portability between different Endian systems. For a discussion of Endian-neutral programming techniques see the IBM Technical report by Jim Gillig, *How to Create Endian Neutral Software for Porta-*

/ *bility*, TR54.837. This information was also published in the October and November 1994 issues of *Dr.*
| *Dobb's Journal*.

3.12.2 Big-Endian and Little-Endian Storage

This section shows examples of the storage of data and instructions in Big-Endian and Little-Endian formats. Miscellaneous information follows:

Miscellaneous

- Figure 7 shows an example of a C language data structure, *s*. The value in each byte is shown in the comments portion of this structure. The C language mapping rules will introduce padding for alignment independent of Endian mode. Table 6 shows this structure mapping as it would appear in memory or on recording media in Big-Endian format with the most significant byte (MSB) first. Table 7 shows the structure mapping in Little-Endian format. These two figures represent data as it would look on I/O storage media. Notice that the one-byte character string data is stored in the specified order, but for all other sizes, the data is stored with the MSB at the first address (n) and the next most significant byte at the next address (n+1) for Big-Endian and with the least significant byte first and MSB last for the Little-Endian mode.

```
struc{
        int     a;      /* 0x11121314                      word */
        doubl   b;      /* 0x2122232425262728              doubleword */
        int     c;      /* 0x31323334                      word */
        char    d[7];   /* 'A', 'B', 'C', 'D', 'E', 'F', 'G'  array of bytes */
        short   e;      /* 0x5152                          halfword */
        int     f;      /* 0x61626364                      word */
} s;
```

Figure 7. Example of a C Structure Showing Values of the Elements

Table 6. Structure s in Big-Endian Order

Address (Hex)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	8	**9**	**A**	**B**	**C**	**D**	**E**	**F**
A000	11	12	13	14				
A008	21	22	23	24	25	26	27	28
A010	31	32	33	34	'A'	'B'	'C'	'D'
A018	'E'	'F'	'G'		51	52		
A020	61	62	63	64				

Table 7. Structure s in Little-Endian Order								
Address (Hex)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	8	**9**	**A**	**B**	**C**	**D**	**E**	**F**
A000	14	13	12	11				
A008	28	27	26	25	24	23	22	21
A010	34	33	32	31	'A'	'B'	'C'	'D'
A018	'E'	'F'	'G'		52	51		
A020	64	63	62	61				

- Figure 8 shows a PowerPC assembly language code fragment which references the structure *s*. Table 8 shows this instruction stream with addresses resolved in Big-Endian format on recording media or memory. Notice that the fields of the instructions are placed within the bytes of a word for readability and not to signify where the actual data is placed. Table 9 shows this instruction stream with addresses resolved in Little-Endian memory or recording media. Notice that the instruction stream has been written with the fields ordered backwards within each word to signify the Little-Endian storage of these instructions. Notice that in both instruction streams the resolved reference addresses for quantities in structure *s* are identical.

```
        li      r5,6
        li      r6,8
loop:
        cmplwi  r5,0
        beq     done
        lbz     r4,d(r5)
        slw     r7,r6,r7
        or      r7,r7,r4
        subi    r5,2
        b       loop
done:
        stw     r7,f
```

Figure 8. Example of an Assembly Language Code Fragment

Table 8. Instructions in Big-Endian Order								
Address (Hex)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	8	**9**	**A**	**B**	**C**	**D**	**E**	**F**
B000	li	r5,	6		li	r6,	8	
B008	loop:	cmplwi	r5,	0	beq	B024		
B010	lbz	r4,	A014	(r5)	slw	r7,	r6,	r7
B018	or	r7,	r7,	r4	subi	r5,	2	
B020	b	B008			done:	stw	r7,	A020

Table 9. Instructions in Little-Endian Order								
Address (Hex)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	8	**9**	**A**	**B**	**C**	**D**	**E**	**F**
B000		6	r5,	li		8	r6,	li
B008	0	r5,	cmplwi	loop:			B024	beq
B010	(r5)	A014	r4,	lbz	r7	r6,	r7,	slw
B018	r4	r7,	r7,	or		2	r5,	subi
B020			B008	b	A020	r7,	stw	done:

3.13 Multiprocessor Considerations

/ The *PowerPC Reference Platform Specification* is intended to also support the implementation of symmetric / multiprocessor (SMP) systems. The main differences between uniprocessor and SMP systems are inter- / processor communications, inter-processor synchronization, I/O interrupt redirection and L2 cache recom- / mendations. Storage access ordering and memory coherence are also important elements of a functional / SMP system. Section A.6, “Symmetric Multiprocessor,” provides a detailed implementation example of an / SMP system.

/ Requirements and recommendations follow:

/ ***Requirements***

/ • Inter-processor synchronization: The *lwarx, stwcx, eieio* and *sync* instructions must be used to ensure / synchronization among processors, when required. See the *The PowerPC Architecture* for details.
/ • TLB synchronization: The TLB Invalidate Entry and TLB Synchronize instructions must be used to / synchronize TLBs among processors. The TLB Invalidate All instruction is optional.
/ • Time Base or Real-Time Clock: A means to synchronize the Time Base or Real-Time Clock among / processors must be provided.
/ • Reconfiguration: The system must provide, as part of the boot process, a means to vary a processor / (and associated dedicated second level cache, if any) into and out of the configuration.
/ • A processor not in the configuration must not be allowed to affect the operation of processors which are / in the configuration.
/ • All processors in the configuration must have equal access to system memory.
/ • All processors in the configuration must have equal access to all I/O devices and adaptors.
/ • All processors in the configuration must be of the same type (e.g. 604) and speed (e.g. 100 MHz).
/ • All L2 caches in the configuration must be of the same type (e.g. in-line, copy-back) and size (e.g. 1 / MB).
/ • If an in-line L2 cache is used, it must support one reservation as defined for the *lwarx* and *stwcx* / instructions.

/ ***Recommendations***

/ • A dedicated, in-line, copy-back L2 cache per processor is strongly recommended.
/ • It is recommended that the I/O subsystem be designed to scale with the number of processors. If the / processors are upgradable, it is recommended that the I/O subsystem have the capability to scale with / relative processor performance as well.
/ • It is recommended that multiprocessing systems contain a minimum of 4 KB of Non-volatile Memory / per processor.

3.13.1 MP Interrupts

Figure 52 in Section A.6, “Symmetric Multiprocessor,” shows one possible implementation of MP interrupts.

Requirements, recommendations, and miscellaneous information follow:

Requirements

- The system must support a total of at least 16 inter-processor and I/O interrupt requests.
- There must be a means to redirect any I/O interrupt request to a specific processor or to all processors.
- There must be a means to allow software to generate an individual interrupt request from any processor to any processor (inter-processor interrupts).
- There must be a means to identify the interrupt source.
- The process of handling an interrupt on one processor must not be allowed to disrupt other processors.

Recommendations

- It is recommended that the hardware abstraction software isolate the operating system from the MP interrupt hardware. One possible abstraction interface is described below.
- One inter-processor interrupt request per processor plus one I/O interrupt request per native I/O device plus at least one I/O interrupt request per add-in card slot are recommended.
- A means to allow software to generate a floating system-wide interrupt request is recommended.
- A means to assign a priority level to each interrupt request is recommended. (Required for AIX.)
- A means to assign a priority level to each processor is recommended. (Required for AIX.)
- A means to allow software to poll for pending interrupts is recommended.
- To reduce the frequency of interrupts and to avoid unnecessarily restrictive requirements for low-latency interrupt handling, it is recommended that I/O devices, particularly high-speed serial devices, be buffered.
- Interrupt sharing is not recommended (except when the devices sharing interrupts also share the same device driver and interrupt priority).
- Inter-processor interrupts should be low-latency.

Miscellaneous

- Intel’s 82489DX Advanced Programmable Interrupt Controller meets all the requirements and the first six recommendations listed above. One 82489DX integrated circuit is used per processor. (Note: while PowerPC based systems can use the 82489DX integrated circuit, the local unit and I/O unit logic cores which make up the 82489DX are not licensed for use within PowerPC processors or within chip sets designed for use with PowerPC processors.)
- A possible interrupt abstraction interface:
  - GetInterruptVector
  - EnableInterrupt
  - DisableInterrupt
  - GenerateInterrupt: Software-generated interrupt.
  - RaiseIRQL: Raises the processor’s priority level.
  - LowerIRQL: Lowers the processor’s priority level.
- Future directions for MP interrupts:
  - Interrupts should be associated with each request, not with a card slot, device adaptor or device. Advantages:
    - Allows software to assign a unique interrupt vector to all requests to a particular device. Each device (e.g. fixed disk, CD-ROM, tape) connected to a SCSI controller could have a unique interrupt vector.
    - Allows software to assign different interrupt priority levels to requests to the same device or to the same device adaptor. Requests associated with high-priority tasks could have higher priority interrupt levels than requests associated with lower-priority tasks.

/ – Elimination of sideband signals (e.g. INTA, INTB) for interrupts. Adding an interrupt enqueue
/ command to the PCI bus, for example, could allow a device (or device adaptor) to set an interrupt
/ pending bit within the interrupt controller without the use of sideband signals. Using the same inter-
/ face to enqueue the interrupt as to transfer the data also eliminates some data coherency concerns by
/ ensuring that data buffers are flushed before the interrupt is enqueued.
/ – Adding interrupt enqueue and interrupt dequeue commands to the PowerPC processor bus (as
/ address-only requests) could allow the interrupt controller to be embedded within each processor
/ chip without requiring additional I/O or incurring the additional latency of a serial interrupt inter-
/ face.

3.14 Alignment Considerations

This section defines the requirements and recommendations for use and support of aligned and unaligned operations. It provides directions which will allow PowerPC Reference Platform systems to run applications and operating systems which have a Power legacy and points out changes in that legacy which will have to be accounted for in these PowerPC based machines. Requirements, recommendations, and miscellaneous information follow:

Requirements

/ • Noncachable operations must be handled by other system components in a manner defined in Table 10.

Recommendations

• Because some unaligned operations may have performance penalties, it is strongly recommended that all PowerPC software be restricted to using size-aligned load/store operations.
/ • It is recommended that noncachable load and store multiple and move assist instructions need not be
/ supported by the components of the system outside of the processor.
/ • It is recommended that floating-point load or store instructions which are not word aligned need not be
/ supported by the components of the system outside of the processor.

Table 10 (Page 1 of 2). Specification for Handling of Alignment

Instructions as Defined in the PowerPC Architecture	**Aligned**		**Unaligned**	
	BE	**LE**	**BE**	**LE**
Fixed-Point Load Instructions	P	P	P	E
Fixed-Point Store Instructions	P	P	P	E
Load and Store with Byte Reversal Instructions	P	P	P	E
Fixed-Point Load and Store Multiple Instructions	D	E	E*	E
Fixed-Point Move Assist Instructions (e.g. String)	D	E	D	E
Storage Synchronization Instructions *lwarx* and *stwcx*	P	P	B	B
Floating-Point Load Instructions	P	P	W*	E

Table 10 (Page 2 of 2). Specification for Handling of Alignment

Instructions as Defined in the PowerPC Architecture	Aligned		Unaligned	
	BE	LE	BE	LE
Floating-Point Store Instructions including the optional instruction *stfiwx*	P	P	W*	E

Note:

- For more information refer to the PowerPC User Instruction Set in *The PowerPC Architecture*, Book I.
- Any unaligned operation which crosses a protection boundary may cause the alignment interrupt to be invoked. Refer to Section 5.5.6, PowerPC Operating Environment, in *The PowerPC Architecture*, Book III.
- Big-Endian (BE); Little-Endian (LE)

B Boundedly undefined or a system alignment error handler invoked by the processor
D Discouraged, but cachable operations are currently supported by the PowerPC processors
E System alignment or system error handler invoked by the processor
P Processed normally
W Word alignment of doublewords is only unaligned case allowed
***** The 601 Processor processes unaligned forms of these instructions, but other processors do not

Miscellaneous

- Size alignment will ensure that applications will run with the highest possible performance on all PowerPC hardware implementations. The degradation of performance is dependent upon the particular processor used and the implementation approach of other hardware system components. Size alignment is defined as follows:

| **Halfword** The address is divisible by 2 and is in the form B'x...xxx0'

| **Word** The address is divisible by 4 and is in the form B'x...xx00'

| **Doubleword** The address is divisible by 8 and is in the form B'x...x000'

/ • Table 10 defines the supported and unsupported forms of the instructions for cachable and noncachable / operations. Note that some processors may have implemented a superset of this support (refer to Section 3.17, “PowerPC Architecture Features Not Recommended”). Cachable operations are handled within the PowerPC processors. Noncachable operations are either 1) cache-inhibited loads and stores to or from memory or 2) loads and stores to or from I/O devices. Notice that some operations are “discouraged.” PowerPC Reference Platform-compliant systems may support these operations as a transition from earlier architectures. Future versions of the PowerPC architecture and processors may not support these instructions. Software which depends upon these instructions may not run on future machines or may suffer a performance impact if these instructions are used.

/ 3.15 Support for Loads and Stores to System I/O Bus

/ This section defines the hardware system support for loads and stores to the system I/O bus. Requirements, recommendations, and miscellaneous information follow:

Requirements

/ • If the processor has an 8-byte data bus, then the system must support all processor-generated loads and / stores to I/O buses for 1-, 2- and 4-byte transfers which do not span a word boundary and 3-byte trans- / fers which do not span a word boundary and are the result of the processor breaking a larger load or / store at the doubleword boundary.

/ • If the processor has a 4-byte data bus, then the system must support all processor-generated loads and
/ stores to I/O buses for 1-, 2- and 4-byte transfers which do not span a word boundary and 3-byte trans-
/ fers which do not span a word boundary and are the result of the processor breaking a larger load or
/ store at the word boundary.
• If a vendor implements a 2-byte bus, then the hardware must accommodate the 4-byte I/O transfers generated on a system. Four-byte transfers would have to be broken into two 2-byte transfers.

Recommendations

• It is recommended that a system not support references which span a word boundary for loads and stores to system I/O buses which are only one word (4 bytes) wide.
/ • It is recommended that a system support 8 byte transfers, if these transfers will result in more efficient
/ utilization of the I/O bus.

Miscellaneous

/ • Table 11 shows the addresses which the processor generates and the required and recommended system
/ responses for I/O to a 4-byte bus. The following information describes this table:
– Combinations of transfer sizes and starting addresses which are presented by the processor to the bus
/ bridge and are required transfers for a word-wide bus are marked as “Support.”
/ – Those transfers which are not recommended for a one-word-wide bus (e.g. transfers which span a word boundary) are marked as “Undefined results.”
– Some combinations of size and starting address will not be generated by the PowerPC processor and are marked as “N/A.” For instance, the PowerPC processors break any request at a doubleword boundary. A word request starting at B'x...x111' becomes a 1-byte transfer from B'x...x111' and a
/ 3-byte transfer from B'x..x1000'. The column of this table labeled “4-byte data bus” represents the
/ actions of a 603 processor. For the 603 and for any future processor with a word-wide data bus, unaligned transfers which cross a word boundary are also broken into two pieces.
– Note that 3-byte transfers occur only as a result of these word and doubleword split transfers or as a result of move assist (e.g. string) instructions. String instructions are not recommended in a PowerPC Reference Platform system. Three-byte transfers due only to string operations which would not cross the word boundary are marked as “undefined results (string).”
– Five-, 6-, and 7-byte transfers are generated only as a result of floating-point doubleword unaligned loads and stores which are implemented only in the 601 processor and are not recommended for PowerPC Reference Platform systems. These transfers are marked as “601 undefined results.”
/ – For any “undefined results,” a system design may provide an error response, may provide undefined data, or may implement support for some or all of these cases. A more complex bus bridge could be designed to buffer requests that spanned a word and send them onto the I/O bus as two requests.
• An 8-byte I/O bus would support the required loads and stores defined in this table. In addition, some “undefined results” conditions in this table such as unaligned words and 8-byte transfers could be handled by the 8-byte bus.
• The terminology used in Table 11 is summarized below:

Term	Meaning
Support	A bus bridge must support this transfer address and size
Undefined Results	A bus bridge may support, may give an error response, or may give undefined data
N/A	A PowerPC processor will not generate this combination of address and size to the PowerPC processor bus
Undefined Results (String)	This combination of address and length will only be produced by a string operation and the bus bridge may support, may give an error message, or may give undefined data

601 Undefined Results This combination of address and length will only be generated from a 601 processor and the bus bridge may support, may give an error response, or may give undefined data

Table 11 (Page 1 of 2). Processor-Generated Load and Store Addresses to System I/O Buses

Transfer Size	CPU Address	Response For System With Processor Which Has	
		8-Byte Data Bus	4-Byte Data Bus
Byte	B'x...x000'	Support	Support
	B'x...x001'	Support	Support
	B'x...x010'	Support	Support
	B'x...x011'	Support	Support
	B'x...x100'	Support	Support
	B'x...x101'	Support	Support
	B'x...x110'	Support	Support
	B'x...x111'	Support	Support
Halfword	B'x...x000'	Support	Support
	B'x...x001'	Support	Support
	B'x...x010'	Support	Support
	B'x...x011'	Undefined Results	N/A
	B'x...x100'	Support	Support
	B'x...x101'	Support	Support
	B'x...x110'	Support	Support
	B'x...x111'	N/A	N/A
3-byte	B'x...x000'	Support	Support
	B'x...x001'	Undefined Results (String)	Support
	B'x...x010'	Undefined Results	N/A
	B'x...x011'	Undefined Results	N/A
	B'x...x100'	Undefined Results (String)	Support
	B'x...x101'	Support	Support
	B'x...x110'	N/A	N/A
	B'x...x111'	N/A	N/A

Table 11 (Page 2 of 2). Processor-Generated Load and Store Addresses to System I/O Buses			
Transfer Size	**CPU Address**	**Response For System With Processor Which Has**	
		8-Byte Data Bus	**4-Byte Data Bus**
Word	B'x...x000'	Support	Support
	B'x...x001'	Undefined Results	Undefined Results
	B'x...x010'	Undefined Results	Undefined Results
	B'x...x011'	Undefined Results	Undefined Results
	B'x...x100'	Support	Support
	B'x...x101'	N/A	N/A
	B'x...x110'	N/A	N/A
	B'x...x111'	N/A	N/A
5-byte	Any	N/A (601 Undefined Results)	N/A
6-byte	Any	N/A (601 Undefined Results)	N/A
7-byte	Any	N/A (601 Undefined Results)	N/A
/ Doublewords	B'x...x000'	Undefined Results*	Undefined Results*
	B'x...x001'	N/A	N/A
	B'x...x010'	N/A	N/A
	B'x...x011'	N/A	N/A
	B'x...x100'	N/A	N/A
	B'x...x101'	N/A	N/A
	B'x...x110'	N/A	N/A
	B'x...x111'	N/A	N/A
/ **Note:** * Recommend support, if more efficient utilization of the bus will result.			

3.16 Cache-Inhibited Loads and Stores to System Memory

/ This section describes the hardware system support for cache-inhibited loads and stores to System Memory. Requirements, recommendations and miscellaneous information follow:

Requirements

/ • If the processor has an 8-byte data bus, then the system must support all processor-generated loads and
/ stores to system memory for 1-, 2-, 4- and 8-byte transfers which do not span a doubleword boundary
/ and 3-byte transfers which do not span a doubleword boundary and are the result of the processor
/ breaking a larger load or store at the doubleword boundary.
/ • If the processor has a 4-byte data bus, then the system must support all processor-generated loads and
/ stores to system memory for 1-, 2- and 4-byte transfers which do not span a word boundary; 3-byte
/ transfers which do not span a word boundary and are the result of the processor breaking a larger load
/ or store at the word boundary; and if the processor is in 8-byte mode, 8-byte transfers which do not
/ span a doubleword.

Recommendations

- It is recommended that systems not support references which span a doubleword boundary for cache-inhibited loads and stores to system memory.

Miscellaneous

/ • Table 12 shows the addresses which the processors generate and the required response of the system for
/ loads or stores to system memory. The following describes this table:
  – Combinations of transfer sizes and starting addresses which are presented by the processor to the
/ memory controller and are required transfers are marked as “Support.”
  – Some combinations of transfer size and starting address will not be generated by the PowerPC processor and are marked as “N/A..” For instance, the PowerPC processors break any request at a doubleword boundary. A word request starting at B'x...x111' becomes a 1-byte transfer from
/ B'x...x111' and a 3-byte transfer from B'x..x1000'. The column in this table marked “4-byte data
/ bus” represents the actions of the 603 processor. For the 603 and for any future processor with a word-wide data bus, unaligned transfers which cross a word boundary are also broken into two pieces.
  – Note that 3-byte transfers occur only as a result of these word and doubleword split transfers or as a result of move assist (e.g. string) instructions which are not recommended in a system. Three-byte transfers which are generated only as a result of string operations are marked as “Undefined results (string).”
  – Five-, 6-, and 7-byte transfers are generated only as a result of doubleword unaligned loads and stores which are implemented only in the 601 processor. Unaligned Floating-Point loads and stores are not recommended for systems and are marked as “601 undefined results.”
  – For any “undefined results,” an implementation may provide an error indication, may provide undefined data, or may support these transfers.
- The terminology used in Table 12 is summarized below:

Term	Meaning
Support	A memory controller must support this transfer address and size
N/A	A PowerPC processor will not generate this combination of address and size to the PowerPC processor bus
Undefined Results (String)	This combination of address and length will only be produced by a string operation and the memory controller may support, may give an error message, or may give undefined data
601 Undefined Results	This combination of address and length will only be generated from a 601 processor and the memory controller may support, may give an error response, or may give undefined data

Table 12 (Page 1 of 2). Cache-Inhibited Load and Store Addresses to System Memory

Transfer Size	CPU Address	Response For System With Processor Which Has	
		8-Byte Data Bus	4-Byte Data Bus
Byte	B'x...x000'	Support	Support
	B'x...x001'	Support	Support
	B'x...x010'	Support	Support
	B'x...x011'	Support	Support
	B'x...x100'	Support	Support
	B'x...x101'	Support	Support
	B'x...x110'	Support	Support
	B'x...x111'	Support	Support
Halfword	B'x...x000'	Support	Support
	B'x...x001'	Support	Support
	B'x...x010'	Support	Support
	B'x...x011'	Support	N/A
	B'x...x100'	Support	Support
	B'x...x101'	Support	Support
	B'x...x110'	Support	Support
	B'x...x111'	N/A	N/A
3-byte	B'x...x000'	Support	Support
	B'x...x001'	Undefined results (string)	Support
	B'x...x010'	Undefined results (string)	N/A
	B'x...x011'	Undefined results (string)	N/A
	B'x...x100'	Undefined results (string)	Support
	B'x...x101'	Support	Support
	B'x...x110'	N/A	N/A
	B'x...x111'	N/A	N/A
Word	B'x...x000'	Support	Support
	B'x...x001'	Support	N/A
	B'x...x010'	Support	N/A
	B'x...x011'	Support	N/A
	B'x...x100'	Support	Support
	B'x...x101'	N/A	N/A
	B'x...x110'	N/A	N/A
	B'x...x111'	N/A	N/A
5-byte	Any	N/A (601 Undefined results)	N/A
6-byte	Any	N/A (601 Undefined results)	N/A

Table 12 (Page 2 of 2). Cache-Inhibited Load and Store Addresses to System Memory			
Transfer Size	**CPU Address**	**Response For System With Processor Which Has**	
		8-Byte Data Bus	**4-Byte Data Bus**
7-byte	Any	N/A (601 Undefined results)	N/A
/ Doublewords	B'x...x000'	Support	Support*
	B'x...x001'	N/A	N/A
	B'x...x010'	N/A	N/A
	B'x...x011'	N/A	N/A
	B'x...x100'	N/A	N/A
	B'x...x101'	N/A	N/A
	B'x...x110'	N/A	N/A
	B'x...x111'	N/A	N/A
Note: * N/A when the 603 is running in 4-byte data bus mode			

3.17 PowerPC Architecture Features Not Recommended

Some PowerPC Architecture features need not be implemented in a PowerPC Reference Platform-compliant system. Some of these features are transition or carryover from an earlier Power architecture. Other of these features are not supported consistently in PowerPC implementations. The intent of recommending that PowerPC Reference Platform-compliant systems not use these features is to position these systems for support across the full family of emerging processors. The features which are not recommended for implementation in a PowerPC Reference Platform-compliant system are described below.

3.17.1 Unaligned Little-Endian Scalar Operations

Recommendations and miscellaneous information follow:

Recommendations

- Because of performance impacts, software running on PowerPC Reference Platform systems is strongly discouraged from using unaligned Little-Endian scalar operations.

Miscellaneous

- The current PowerPC processors do not support any unaligned Little-Endian load or store operations. The PowerPC processor will interrupt to the system alignment handler for resolution.

3.17.2 Unaligned Little-Endian Multiple Scalar Operations

Recommendations and miscellaneous information follow:

Recommendations

- Because of performance impacts, software running on PowerPC Reference Platform systems is strongly discouraged from using unaligned Little-Endian multiple scalar operations.

Miscellaneous

- The current PowerPC processors do not support any Little-Endian multiple scalar load or store operations. This includes load and store multiple and load and store string operations. The PowerPC processor will interrupt to the system alignment handler for resolution.

3.17.3 Direct-Store Segments

Requirements and miscellaneous information follow:

Requirements

- As a minimum, operating system software and PowerPC Reference Platform systems must support ordinary storage segments.
- PowerPC Reference Platform-compliant operating systems must not depend on direct-store segments
/ when running on PowerPC Reference Platform machines.

Miscellaneous

- Section 12.6 of the *The PowerPC Architecture* defines the architecture of direct-store segments as "a mapping of effective addresses onto an external address space, typically an I/O bus." This capability allows for synchronous I/O to other address spaces. Direct-store segments need not be implemented in PowerPC Reference Platforms. As an option in unique environments tied to specific expansion bus support, some operating systems and some systems may support both ordinary storage segments and direct-store segments.

3.17.4 Load and Store String

Requirements, recommendations, and miscellaneous information follow:

Requirements

- Software for PowerPC Reference Platform implementations must not use the load and store string instructions for cache-inhibited access to either I/O or System Memory.

Recommendations

/ • It is recommended that a memory controller/bus bridge implementation for a system reduce complexity and cost by not supporting the 3-byte transfers generated only by load and store string operations.

Miscellaneous

- Section 3.3 of the *The PowerPC Architecture* defines four instructions which load a string of 1 through n bytes into consecutive registers 4 bytes at a time. 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. Processors currently fully support these instructions. These instructions are not efficient for noncachable operations and are the only instructions which generate 3-byte loads and stores. The specific transfers for the processor to I/O bus and for cache-inhibited accesses of the processor to System Memory are defined in Table 11 and Table 12.

3.17.5 Load and Store Multiple

Requirements, recommendations, and miscellaneous information follow:

Requirements

- Software for PowerPC Reference Platform implementations must not use the load and store multiple instructions for cache-inhibited access to I/O or System Memory.

Recommendations

/ - It is recommended that a memory controller/bus bridge implementation for a system treat cache-inhibited load and store multiple instructions the same as word or doubleword transfers because they appear as a series of word, or in the case of storage gathering, doubleword transfers.

Miscellaneous

- Section 3.3 of the *The PowerPC Architecture* defines the load and store multiple instructions which will
/ move one to 32 words to or from consecutive registers. The architecture states that if the words are not aligned, the system alignment error handler should be invoked or the results might be undefined. As a transition feature, the MPC601 processor allows these instructions to be unaligned and treats them as a series of unaligned word fetches or stores. 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. Processors currently fully support these instructions. Some processors implement “storage gathering” which collects adjacent aligned quantities into single larger bus transfers. In this way, a store multiple to an I/O bus could become a series of doubleword transfers which may not be supported on the bus. The specific transfers for the processor to I/O bus and for cache-inhibited accesses of the processor to System Memory are defined in Table 11 and Table 12.

3.17.6 Unaligned Floating-Point Load and Store

Requirements, recommendations, and miscellaneous information follow:

Requirements

/ - Software for PowerPC Reference Platform implementations must not use the unaligned load and store
/ floating-point doubleword instructions for cache-inhibited access to I/O or System Memory.

Recommendations

- It is strongly recommended that software not use unaligned load and store floating-point operations.
/ - It is recommended that a memory controller/bus bridge implementation for a system reduce complexity
/ and cost by not supporting the 5-, 6- and 7-byte transfers generated only by unaligned load and store
/ floating-point doubleword instructions.

Miscellaneous

/ - The architecture requires word and doubleword floating-point load and store instructions to be word aligned as defined in *The PowerPC Architecture*, Section 13.5.6. As a transition feature, the 601 processor allows these instructions to be unaligned. An unaligned form of a doubleword floating-point noncachable load or store would generate 5-, 6- or 7-byte transfers as the doubleword is broken at the doubleword boundary.

3.17.7 External Control Instructions

Recommendations and miscellaneous information follow:

Recommendations

/ • It is recommended that external control instructions not be supported in a system unless they are
/ exploited by devices which use this form of bus transfer.

Miscellaneous

- The *The PowerPC Architecture*, Appendix A, defines two instructions for problem-state programs to communicate with special purpose devices. These two instructions, *ecowx* and *eciwx*, present unusual PowerPC processor bus signals. They are coded as address-only transfers but have data.

3.17.8 Special Direct-Store Segment

Recommendations and miscellaneous information follow:

Recommendations

/ • It is recommended that software which uses the special direct-store segment feature of the 601 processor
/ encapsulate this usage in a service.
/ • It is recommended that this service support other PowerPC processors that do not have this 601-unique
/ capability.

Miscellaneous

- The 601 processor has implemented a special direct-store segment which is treated differently than other direct-store segments. When a direct-store segment has the Bus Unit Identifier (BUID) set to X'7f', the processor does a special translation to a real address. The load or store is cache inhibited to this address.

4.0 Machine Abstractions

Historically in the PC industry, operating systems have been intertwined with the hardware on which they execute. Vendors made sure that operating systems would run on their platforms by cloning hardware that was known to run these operating systems. While this had the advantage of allowing the PC operating system industry to flourish, it curbed the number of hardware modifications made to PC platforms by hardware manufacturers. Vendors did not want to jeopardize the ability of their platforms to run as many off-the-shelf operating systems as possible.

The advent of abstraction software and microkernel-based operating systems has allowed operating systems to be more portable across different platforms. Abstraction software concentrates operating system hardware-dependent code into a collection of code that has well-defined interfaces with the operating system kernel and may be modified to meet the hardware interface. An operating system uses abstraction software to interface with system components such as processor and system registers, interrupt controllers, and I/O devices. The operating system is buffered from the hardware of a platform. Thus, moving an operating system to another binary-compatible platform now implies porting only the abstraction software of that operating system to the new platform. A hardware system vendor with a differentiated system would have to supply replacement abstraction software which bridged the gap between the distributed operating system with its standard set of abstractions and the differentiated hardware. This abstraction approach reduces time to market and allows operating system vendors to support a single version of the operating system. Similarly, microkernel-based operating systems concentrate hardware-dependent code and kernel services into a collection of code that is separate from the OS "personality."

The *PowerPC Reference Platform Specification* has been created to utilize these abstraction processes to allow hardware differentiation. It allows vendors to design unique hardware platforms that use off-the-shelf operating systems. Hardware platforms and operating systems that meet this specification are termed "PowerPC Reference Platform compliant." To enable the same operating system to run on differentiated PowerPC Reference Platform-compliant hardware, the PowerPC Reference Platform Specification requires that compliant operating systems be designed to use abstraction software to interface to the hardware. For an operating system to be PowerPC Reference Platform compliant, it must have the qualities described below:

- The operating system must provide software abstractions for the functions described in the subsequent subsections of this chapter.
- The operating system must provide a mechanism to allow the abstraction software to be replaced by other vendors.
- The operating system must provide a mechanism to allow the replacement abstraction software to be merged with the distributed operating system and to run with that operating system.
- The operating system abstraction process must not require access and recompilation of portions of the operating system outside the abstraction software.

An operating system vendor may choose to port to a PowerPC Reference Platform-compliant hardware implementation, but may choose not to meet the above requirements. This approach will limit broad support of that operating system. These operating systems are not PowerPC Reference Platform compliant.

This specification encourages PowerPC based platform vendors to examine the use of abstraction software and microkernel-based operating systems on their platforms. At present, not all operating systems have these architectures. However, many operating systems vendors are migrating to this approach, so it is important that platform vendors be aware of the new technology.

Note: The abstraction requirements in this document are meant to enhance portability of operating systems and device drivers across PowerPC Reference Platforms. They are not intended to insure portability to non-compliant platforms. For that, other abstractions may be necessary.

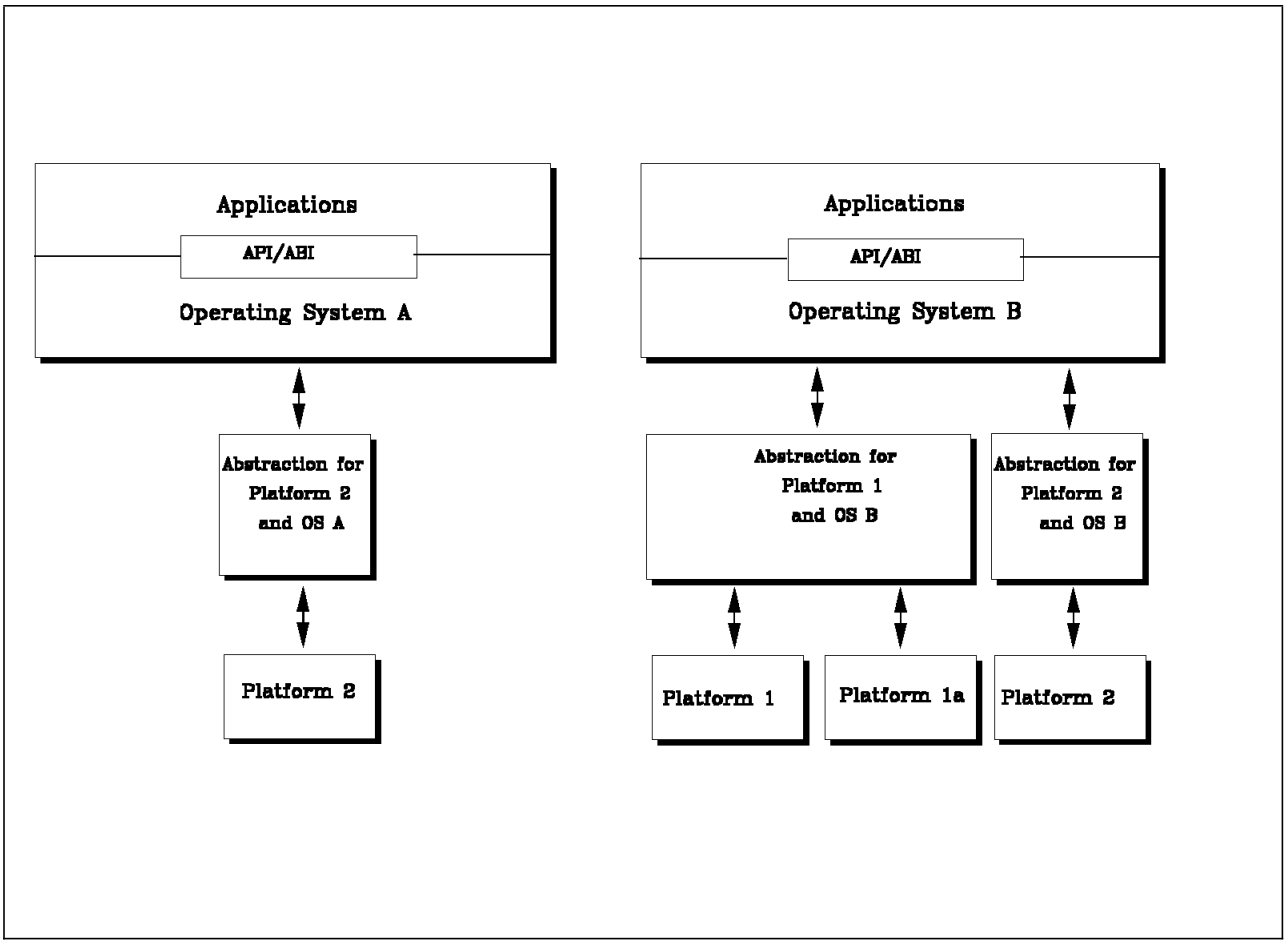


Figure 9. Abstraction Software for Various Platforms

4.1 Abstraction Example

The left side of Figure 9 demonstrates that support for an operating system which runs on Platform 2, called Operating System A, is provided by a set of software abstractions. The right side of the figure shows that a different set of abstraction software is used to support Operating System B. Note that the abstraction software is different among PowerPC Reference Platform-compliant systems that support this platform. The PowerPC Reference Platform defines the functions which must be abstracted, but it does not define the interface to the operating system nor does it define the way the functions are collected into usable services.

In this example, if Platform 1 has hardware that is different than Platform 2, another implementation of the
/ abstraction software must be supplied to support execution of Operating System B. The hardware vendor
/ would supply the abstraction software that allows B to run on Platform 1. Platform 1a is the same system as Platform 1 with an upgraded processor, or a clone of Platform 1 manufactured by a different vendor. Since the hardware is similar enough to support use of the same abstraction software, the operating system will run on both platforms without another implementation of abstraction software.

4.2 Abstraction Software Components

The following subsections describe the abstraction software components. These components are shown in Figure 10 and consist of the Boot-Time Abstraction Software (BTAS) and the Run-Time Abstraction Software (RTAS). This diagram shows a logical collection of abstraction software and is not intended to show an implementation approach. For example, a PowerPC Reference Platform-compliant operating system may implement these functions as a replaceable layer, as well-defined overlayable components of the kernel, or as a small, replaceable kernel. Operating system vendors can use the information in this section as guidance in determining if their operating system adheres to the abstractions that the PowerPC Reference Platform is promoting. Hardware vendors can gain from this information a general idea of what is required to port a PowerPC Reference Platform-compliant operating system to their platforms. Hardware vendors should also refer to specific abstraction software documents provided by operating system vendors for operating systems they wish to port.

4.3 Boot-Time Abstraction Software

The BTAS is a collection of firmware and software which abstracts the hardware that a platform's boot program (e.g. firmware) uses at boot time. It also abstracts the hardware that the operating system loader uses to load an operating system. A hardware platform vendor that provides boot hardware different than that expected by the operating system loader and boot firmware must supply replacement components for
/ the BTAS or must provide other methods of using the new boot devices, such as Open Firmware. Examples of the hardware that the BTAS must abstract are devices such as ASCII terminals, graphics monitors, and keyboards. These devices allow the loader to interact with a user during the loading of the operating system. The BTAS must abstract mass storage and network devices so that the operating system binary can be loaded.

4.4 Run-Time Abstraction Software

The RTAS is a collection of data and software that abstracts hardware from the operating system kernel.
/ The RTAS is made up of system abstractions and device drivers. Some system abstractions may be used to
/ abstract device drivers from hardware. Examples of items that the RTAS abstracts are interrupt controllers and cache configuration.

The software that implements the RTAS is unique for each combination of operating system and differentiated platform hardware. Initial versions of the RTAS are distributed by an operating system vendor. The distributed RTAS is written for one or more hardware platforms. Vendors who differentiate their hardware platforms must make sure that the RTAS supports their hardware. If not, they must develop and distribute replacement components for the RTAS.

This section specifies a minimum set of hardware features that the RTAS must abstract. Equivalently, it specifies the hardware features that a vendor may change and then have RTAS functions defined to bridge the gap.

4.4.1 Data

The RTAS contains areas of data used to store and pass system configuration information from the boot process to the operating system. This configuration information includes processor parameters, memory map information, system bus information, and I/O device information. These areas are:

- the NVRAM area
- the Residual Data area

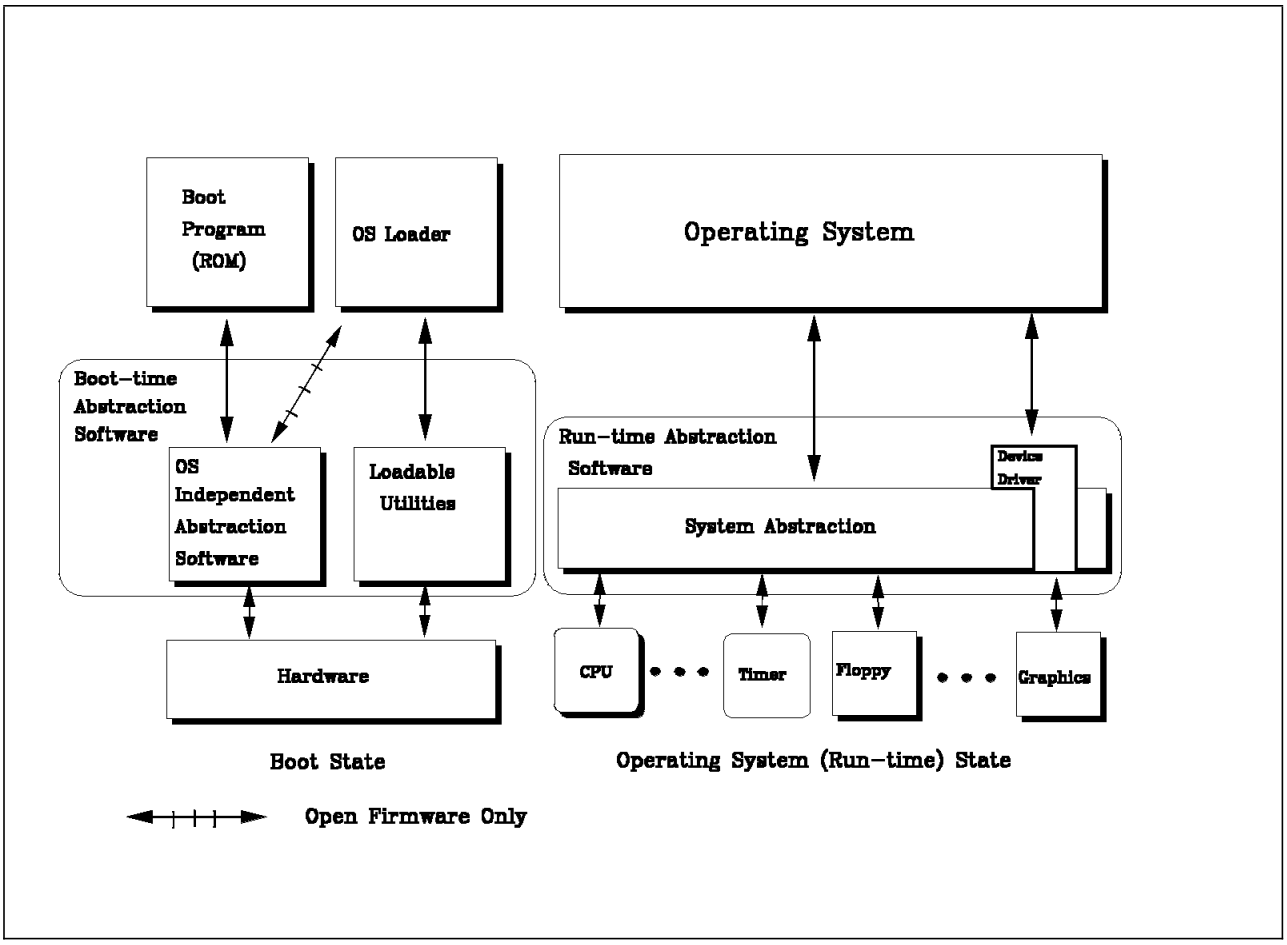


Figure 10. Software Abstraction Layers

4.4.1.1 System Information

The RTAS must provide information about each processor in the system. It must also provide the starting addresses and lengths of each distinct area in the memory map.

4.4.1.2 System Memory

The RTAS must provide information about System Memory. This includes the type, size, and physical address ranges of each contiguous System Memory area that is available or unavailable to the operating system. Any special attributes that are associated with these areas of System Memory must also be provided.

4.4.1.3 I/O Device Information

The RTAS must provide information about a platform's I/O devices. This must include the types, addresses, and quantity of I/O devices and I/O buses.

4.4.2 Processor Initialization

The RTAS must provide services to perform any processor initialization not performed by the operating system loader. For operating systems which support multiprocessors, the RTAS must provide services to locate and initialize other processors.

4.4.3 Flushing of Temporary I/O Buffers

/ If an operating system supports software-managed coherency, then the RTAS must provide services to main-
/ tain coherency by flushing any hardware-provided temporary I/O buffers which are not automatically flushed
/ after a transfer.

4.4.4 Virtual Memory Management

The RTAS must encapsulate the data structures and operations associated with virtual memory management
/ support. These abstractions would support differences within processors designed to comply with *The*
/ *PowerPC Architecture*, Books I, II and III. Examples of the type of abstractions required are described in
/ the next two subsections.

4.4.4.1 TLB Flush

The PowerPC architecture does not specify the size or organization of TLBs, nor does it require the hardware to automatically maintain a TLB coherent with memory-based page tables. Therefore, the operating system must manage the TLBs. The RTAS must provide TLB flush routines that map the TLB flush primitives required by the operating system to the appropriate set of TLB invalidation instructions provided by the specific processor.

4.4.4.2 TLB Reload

Some PowerPC processors provide automatic reload of TLB entries from memory-based page tables. However, some processors do not implement this mechanism and instead rely on a hardware-assisted software routine to refill the TLB. The RTAS must provide services to load a TLB entry in a manner consistent with the processor(s) of a platform.

4.4.5 Cache Management

The RTAS must manage the caches in the system. PowerPC processors may have different sizes of cache and a single cache may combine data and instructions or the processor may have separate caches for data and instructions. In addition, PowerPC Reference Platform systems may have external caches which are write-through or copy-back.

4.4.6 Interrupt Handling

The RTAS must provide services to acknowledge the interrupts and to enable and disable interrupts from devices used on the platform.

4.4.7 Direct Memory Access (DMA)

The abstraction layer must provide abstractions for any DMA devices which are independent of a particular device, such as those built in as inseparable components of the platform. If a particular device (e.g. a disk controller) has a built-in DMA controller, that DMA controller would be controlled by the device driver.
/ Abstraction software services which might be required depend on the particular hardware support. Typical
/ services are as follows:

- Block transfers from memory to the I/O adaptor
- Block transfers from the I/O adaptor to memory
- Byte and halfword bus transfer sizes
- Block transfers of large sizes
- Scatter/gather with single-byte resolution

- Start DMA transfer
- Stop DMA transfer indication of DMA transfer completion (e.g. interrupt or read DMA counter)
- Flush intermediate DMA transfer buffers

Additional features such as generalized memory-to-memory transfer, word transfer sizes, additional DMA channels and chained DMA may be provided by the platform hardware. The abstraction software may provide support for these services.

4.4.8 Calendar and Timer Services

These run-time abstractions provide services for the various clocks and calendars within the system. They must account for the non-volatile Real-Time Clock in the system, the 601 processor RTC, the Time Base on other PowerPC processors, the decrementer which provides an interrupt after counting down to zero, and
| the ability to change clock frequency supplied on some PowerPC processors. Operating Systems which plan
| to participate in multiboot scenarios must maintain the non-volatile Real-Time Clock in GMT.

4.4.9 I/O Addresses

The RTAS must provide a service which allows the device drivers to determine the physical address of a device.

4.4.10 Power Management

/ If macro power management is supported by the operating system, then the abstraction layer must provide a means to change device and subsystem power states. It must also provide a means to read and write system information.

4.4.11 Hardware Fault

The abstraction layer must provide a means for notifying the operating system that a hardware fault has occurred. The RTAS must provide support as follows:

- It must provide a means for notifying the operating system that a memory error has occurred
- It must provide a means for notifying the operating system that an I/O device error has occurred
- It must provide a means for notifying the operating system that a bus timeout error has occurred

4.4.12 Device Drivers

Device drivers are a part of the RTAS. Device drivers are normally distributed by hardware device vendors. They meet a defined interface at the operating system and perform hardware-device-specific operations. To make the device drivers more portable it is strongly recommended that device drivers for PowerPC Reference Platform-compliant systems:

- use the RTAS services provided by each operating system to make the device drivers platform independent
- be written in an Endian-aware manner to allow the driver to be easily ported to Big-Endian and Little-Endian operating systems

/ PCMCIA Socket Services are normally provided by the hardware system vendor. If an operating system
/ provides them, then the operating system must also provide a method for hardware vendors to supply alter-
/ nate Socket Service device drivers.

5.0 Boot Process and Firmware

This section describes the boot process, the format and contents of boot information, and the state of the / system at the end of the boot process. In this section, the required features for PowerPC Reference Platform / systems are stated in terms of "must." The features that are used to improve usability or performance are / described in terms of "recommended."

/ It is a goal of the PowerPC Reference Platform developers to implement IEEE standard P1275 for Boot / Firmware, "Open Firmware." Systems delivered after June 1, 1995, must implement "Open Firmware" to / be compliant with the *PowerPC Reference Platform Specification*. Appendix I, "PowerPC Supplement to IEEE 1275," defines the extension of Open Firmware for a system based on the PowerPC microprocessor. This section describes Open Firmware for the PowerPC Reference Platform.

| **Note:** In the following discussion, firmware that is not compliant to the Open Firmware standard is | referred to as "conventional firmware."

The firmware for a PowerPC Reference Platform-compliant machine must load into memory the load image, which is to take control of the machine. As part of that process, the firmware must initialize some of the devices on the system. The firmware may run diagnostics on the hardware on which it depends, but the loaded operating systems must not assume that complete diagnostics have been run.

Common sources for the load image include disk, diskette, CD-ROM and network connection. Depending upon the type of PowerPC Reference Platform system being produced and the requirements of the operating system being hosted, all of these devices do not need to be supported on all machines. For instance, a medialess machine might require only the ability to boot from the network.

The steps involved in the boot process are shown in Figure 11. Upon power-on, the firmware stored within the system ROM must initialize enough hardware to load the load image. This initialization results in the *cold-start transient state* of the system. The firmware then loads the boot record that contains data structures defining the location of the load image. Next, the firmware loads the load image into memory. Finally, the firmware transfers control to the entry point of the load image, thus concluding the original transient system state. The code in the load image establishes whatever state it requires to proceed with its function. The subsections below describe this process in more detail and define formats and data structures necessary for the boot process.

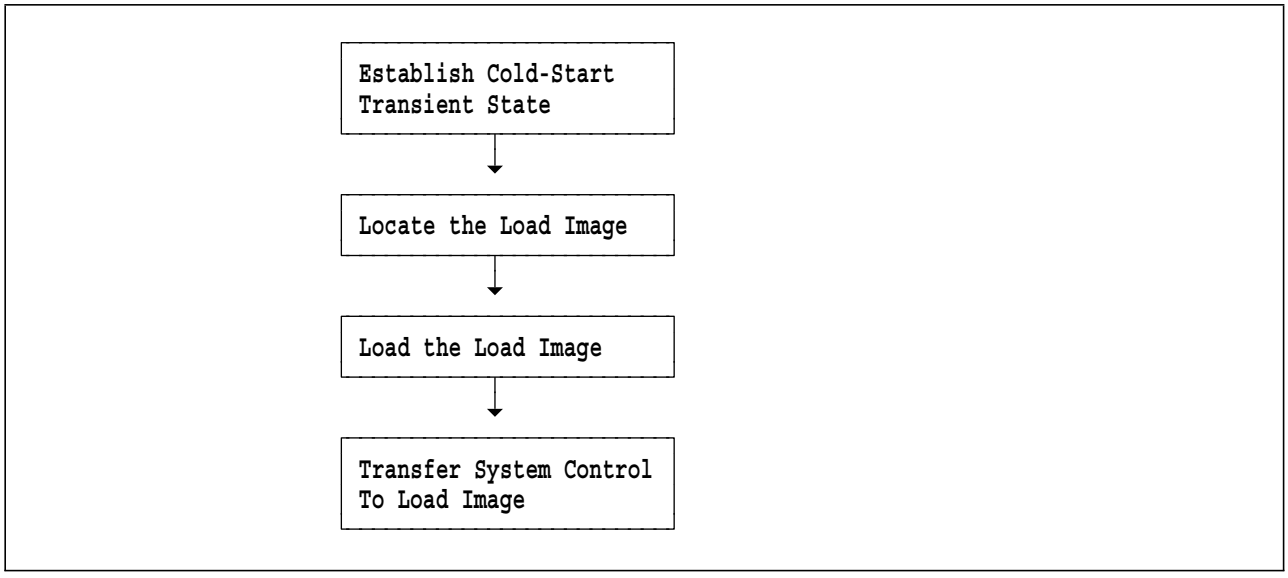


Figure 11. Boot Process Overview

5.1 Establishing Cold-Start Transient State

The boot process must prepare enough of the system to execute boot activities, and must discover the existence of devices with boot images. Minimum tasks to be done in this process are as follows:

a) Power-On Self Test (POST)
b) Configuring the console
c) Obtaining the user's password (if required)
/ d) Configuring the system for boot

5.1.1 Power-On Self Test (POST)

The PowerPC Reference Platform firmware must check the critical core components (the components of the system necessary to successfully complete the boot process). The processor must be checked to see that it is functioning. Enough memory to run the boot must be initialized and tested. A check such as a Cyclic Redundancy Check (CRC) may be performed on the contents of the System ROM to ensure that uncorrupted code exists on the ROM.

5.1.2 Configuring the Console

It is recommended that firmware locate and initialize the console as early as possible to allow the display of progress messages and error messages. In the event of error during the boot, the message must suggest the user's next action. This action might be to try another boot device, to run diagnostics, or to try a power off and on cycle. It is strongly recommended that the message be configurable to either the natural language of
/ a user or language-independent interfaces such as icons.

The primary console may be located by using data stored in NVRAM, or by probing the likely console locations. If more than one console is located, the firmware must have a predetermined policy to decide which of the consoles is used in the boot process.

After the console is discovered, it must be initialized and its driver must be made available to firmware. It is recommended that any hardware cursor or mouse-style pointer be turned off until input is requested to avoid distracting the user during the boot process.

5.1.3 Obtaining the User's Password

At least two levels of passwords must be supported on PowerPC Reference Platform-compliant machines. One level is intended for the normal user and allows normal computational use of the system. The second level allows a user the privileges of changing and defining the system configuration as well as all the normal privileges.

If passwords have not been enabled for the particular machine being booted, this step is skipped. If passwords are enabled, the console must request the user's password and the boot process must authenticate it before proceeding with the boot or configuration.

5.1.4 Configuring the System

/ The boot process must configure the components necessary to perform the boot. The rest of the configura-
/ tion process may be deferred to the hosted operating system that will complete the configuration process. If the configuration data in NVRAM identifies the boot devices, the boot process may proceed to check the boot devices. Otherwise, the boot process must check likely boot device candidates. For instance, for the Reference Implementation the firmware checks the diskette drive and devices attached to the SCSI controller. For network boot, software specific to the adaptor will have to be included in the firmware.

The firmware must provide a mechanism that allows a user to perform manual configuration. For example, a user may have the capability to escape from the auto-configuration process and then either define the boot device or redefine the default order of looking at devices. For network boot, the user may be able to set the network identification for the boot image.

5.2 Locating the Load Image

/ The next step in the boot process is to locate the load image in a boot partition. A boot device must have
/ at least one boot partition. The default boot partition and load image are specified by using Global Envi-
/ ronment variables defined in NVRAM. In the normal booting process, firmware uses the default values to
/ select the load image.

/ The firmware must provide a mechanism that allows a user to perform manual selection. For example,
/ when the default device and partition are not specified or a user wants to select other than the default, a user
/ may have the capability to escape from the default booting process to specify the boot device and the
/ location of the load image.

/ The boot record can be used by firmware to identify the possible boot partitions in a device that has mul-
/ tiple partitions. A hard disk must have a boot record. However, a CD-ROM may not have a boot record.
/ In this case, a CD-ROM must be treated as if it has single partition, and the format of the CD-ROM must
/ conform to the ANSI/NISO/ISO 9660 standard, “Information processing -- volume and file structure of
/ CD-ROM for information interchange,” and the load image must be identified by using an ANSI/NISO/ISO
/ 9660-standard file name.

/ **Note:** The ANSI/NISO/ISO 9660 standard specifies the volume and file structure of compact read-only
/ optical disks (CD-ROM) for the interchange of information between users of information processing
/ systems. The PowerPC Open Firmware specification requires compliant firmware to support the
/ ANSI/NISO/ISO 9660 file system for CD-ROM.

/ A diskette device is treated as if it has a single partition.

In the following sections, the structures of the boot record are presented.

5.2.1 Boot Record

/ The format of the boot record is an extension of the PC environment. The boot record is composed of a PC compatibility block and a partition table. To support media interchange, the PC compatibility block may contain an x86-type program. The entries in the partition table identify the PowerPC Reference Platform boot partition and its location in the media.

The layout of the the boot record must be designed as shown in Figure 12. The first 446 bytes of the boot record contain a PC compatibility block, the next four entries contain a partition table totalling 64 bytes, and the last 2 bytes contain a signature.

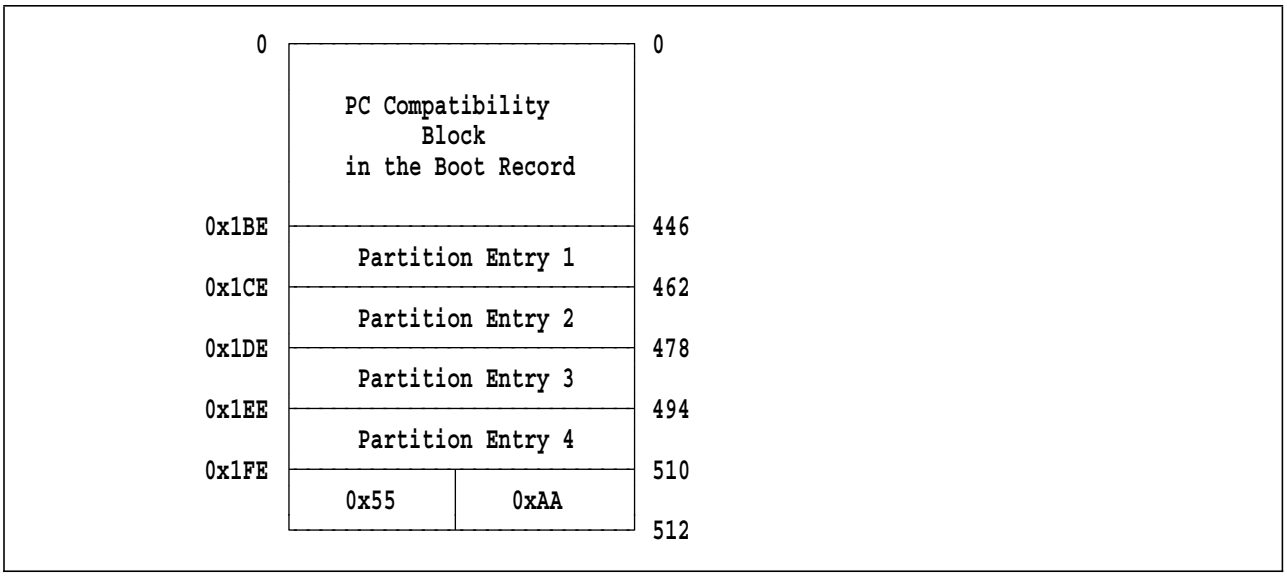


Figure 12. Boot Record -- Detail View

5.2.1.1 PC Partition Table Entry

To support media interchange with the PC, the PowerPC Reference Platform defines the format of the partition table entry based on the PC format. This section describes the format of the PC partition table entry.

A partition table entry occupies 16 bytes. The layout of a PC partition table entry is shown in Figure 13.

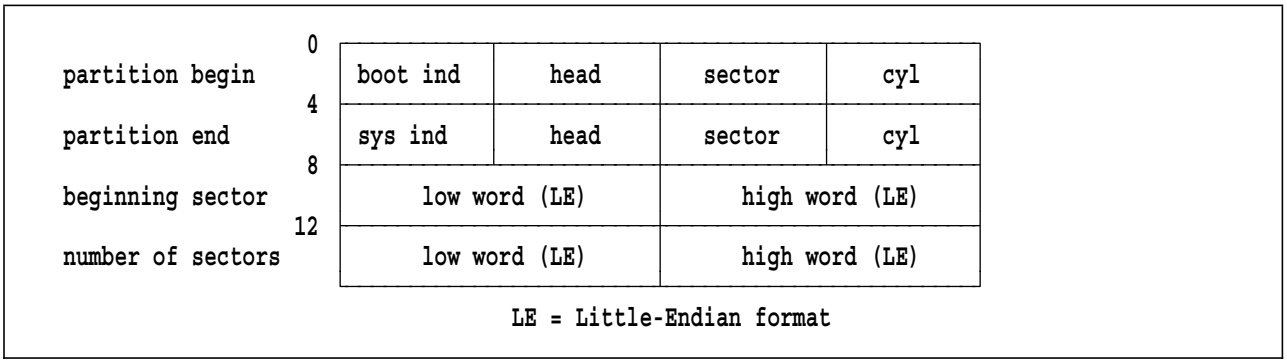


Figure 13. Partition Table Entry

The primary fields of a partition table entry are defined as follows:

partition begin This field contains the beginning address of the partition in head, sector, cylinder notation.

partition end This field contains the end address of the partition in head, sector, cylinder notation.

beginning sector The number of sectors preceding the partition on the disk (that is, the zero-based relative block address of the first sector of the partition).

number of sectors The number of sectors allocated to the partition.

The subfields of a partition table entry are defined as follows:

boot ind Boot Indicator. This byte indicates if the partition is active. If the byte contains 0x00, then the partition is not active and will not be considered as bootable. If the byte contains 0x80, then the partition is considered active.

head An eight-bit value, zero-based.

sector A six-bit value, one-based. The low-order six bits are the sector value. The high-order two bits are the high-order bits of the 10-bit cylinder value.

cyl Cylinder. The low-order eight-bit component of the 10-bit cylinder value (zero-based). The high-order two bits of the cylinder value are found in the sector field.

sys ind System Indicator. This byte defines the type of the partition. There are numerous partition types defined. For example, the following list shows several:

0x00 Available partition
0x01 DOS, 12-bit FAT
0x04 DOS, 16-bit FAT
0x05 Extended DOS partition

/ The extended DOS partition is used to allow more than four partitions in a device. The boot record in the / extended DOS partition has a partition table with two entries, but does not contain the code section. The / first entry describes the location, size and type of the partition. The second entry points to the next partition / in the chained list of partitions. The last partition in the list is indicated with a system indicator value of / zero in the second entry of its partition table.

/ Because of the DOS format limitations for a device partition, a partition that starts at a location beyond the / first 1 gigabyte is located by using an enhanced format shown in Figure 14.

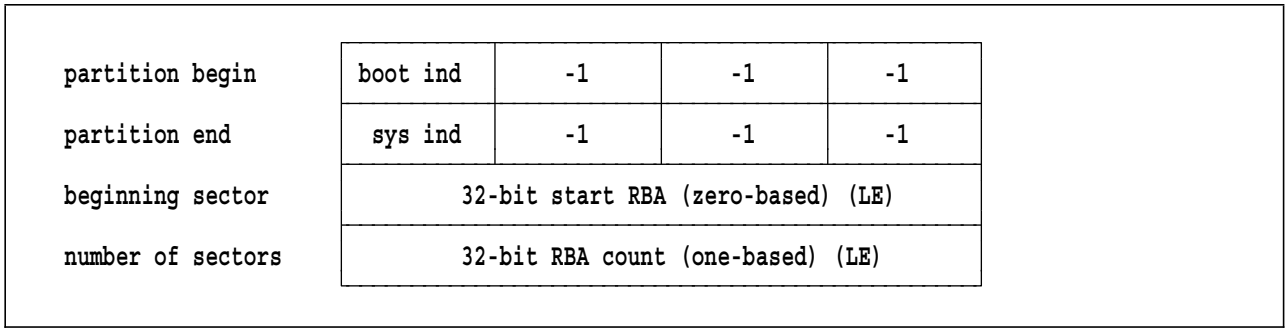


Figure 14. Partition Table Entry Format for an Extended Partition

The value "-1" indicates that the field is all ones. "RBA" means Relative Block Address in units of 512 bytes.

/ 5.2.1.2 PowerPC Reference Platform Partition Table Entry for Conventional Firmware

/ This section describes the definition of the partition table entries for conventional firmware. The definition / of the boot record and its process for Open Firmware are defined in Appendix I, "PowerPC Supplement to / IEEE 1275."

/ Conventional firmware identifies the PowerPC Reference Platform partition table entry (refer to Figure 15) / by the 0x41 value in the system indicator field. All other fields are ignored by the firmware except for the / "beginning sector" and "number of sectors" fields. The "head," "sector," and "cyl" fields must contain / PC-compatible values (i.e. acceptable to DOS) to avoid confusing PC software. These fields, however, may be ignored by the PowerPC Reference Platform firmware. See Section 5.2.1.1, "PC Partition Table Entry," for descriptions of these fields.

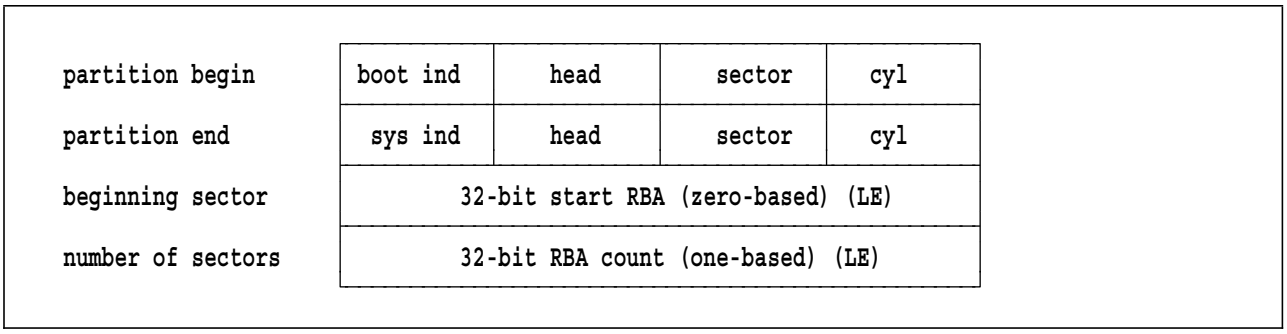


Figure 15. PowerPC Reference Platform Partition Table Entry Format for Conventional Firmware

The 32-bit start RBA is zero-based. The 32-bit count RBA value is one-based and indicates the number of 512-byte blocks. The count is always specified in 512-byte blocks, even if the physical sectoring of the target device is not 512-byte sectors.

5.3 Loading the Load Image

/ The next step in the boot process is to transfer the load image into System Memory. This section describes / the layout of the 0x41 type partition and the process of loading the load image. The structure and the / process of the load image for Open Firmware are described in Appendix I, “PowerPC Supplement to IEEE / 1275.”

/ **Note:** The 0x41-type partition is supported by conventional firmware. An Open Firmware implementation / may support the 0x41 type partition for compatibility. However, after June 1 1995, hardware system and / operating system vendors must format the boot devices as described in the PowerPC Open Firmware specifi- / cation in Appendix I, “PowerPC Supplement to IEEE 1275.”

/ The layout for the 0x41 type partition is shown in Figure 16. The PC compatibility block in the boot partition may contain an x86-type program. When executed on an x86 machine, this program displays a message indicating that this partition is not applicable to the current system environment.

/ The second relative block in the boot partition contains the entry point offset, load image length, flag field, / operating system ID field, ASCII partition name field and the reserved1 area. The 32-bit value entry point / offset (Little-Endian) is the offset (into the image) of the entry point of the PowerPC Reference Platform / boot program. The entry point offset is used to allocate the reserved1 space. The reserved1 area from offset / 554 to Entry Point - 1 is reserved for implementation-specific data and future expansion.

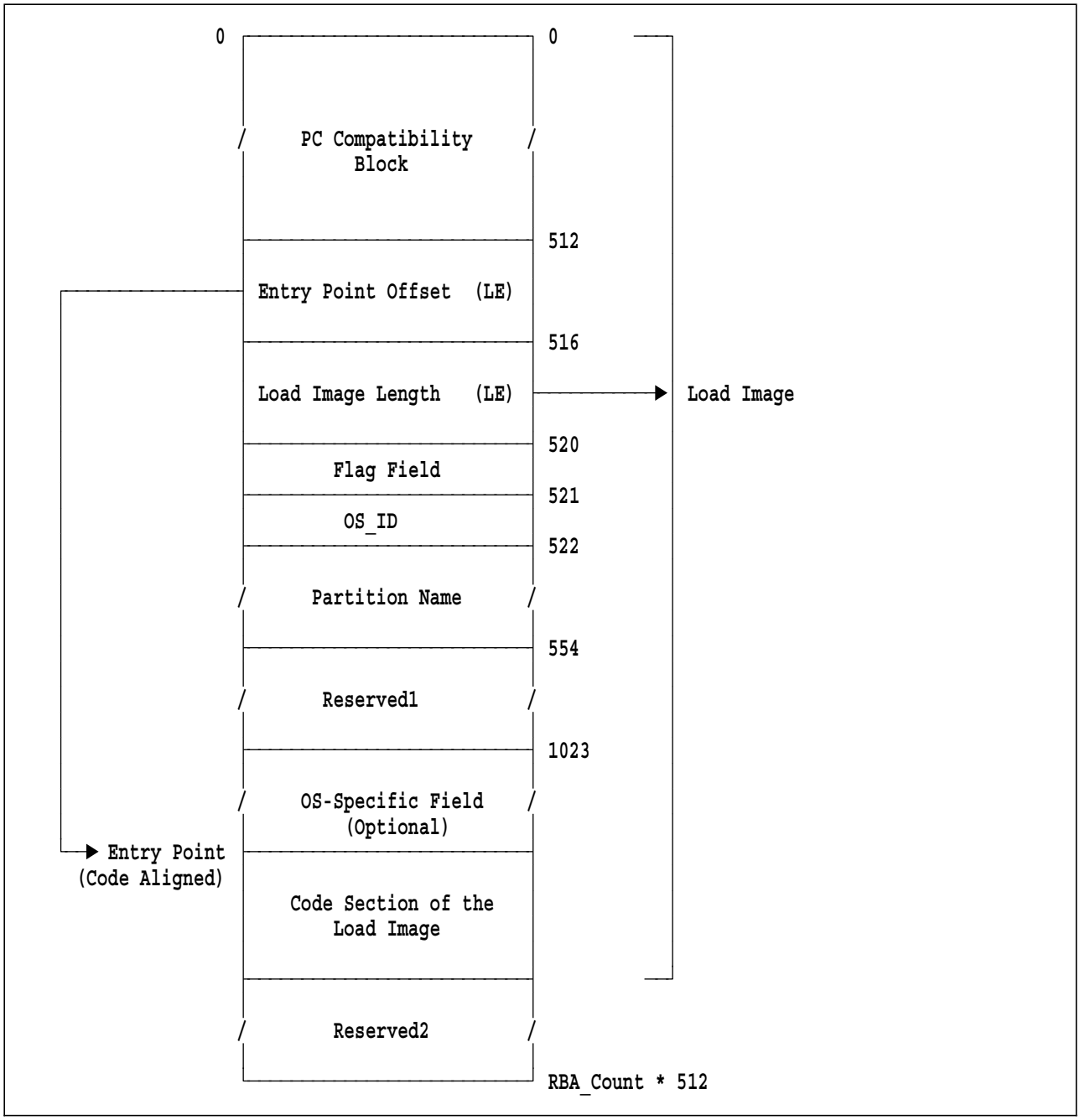
/ The 32-bit value load image length (Little-Endian) is the length, in bytes, of the load image. The load image / length specifies the size of the data physically copied into the system RAM by the firmware.

/ The flag field is 8 bits wide. The MSb in the field is allocated for the Open Firmware flag. If this bit is set to / 1, the loader requires Open Firmware services to continue loading the operating system.

/ The second MSb is the Endian mode bit. If the mode bit is 0, the code in the section is in Big-Endian / mode. Otherwise, the code is in Little-Endian mode. The implication of the Endian mode bit is different / depending on the Open Firmware flag. If the Open Firmware flag is on, the mode bit indicates the Endian / mode of the code section pointed to by the load image offset, and the firmware has to establish the hardware / Endian mode according to this bit. Otherwise, this bit is just an informative field for firmware.

/ The OS_ID field and partition name field are used to identify the operating system located in the partition. / The OS_ID field has the enumerated identification value of the operating system located in the partition. / The 32 bytes of partition name field must have the ASCII notation of the partition name. The name and

/ OS_ID can be used to provide to a user the identification of the boot partition during the manual boot
/ process.



/ Figure 16. Layout of the 0x41-Type Partition

Once the boot partition is located by using the boot record, the firmware will typically:

a) read into memory the second 512-byte block of the load image
b) determine the load image length, which runs up to, but does not not include, the reserved2 space
c) allocate a buffer in system RAM for the load image transfer (no fixed location)
d) transfer rest of the load image into system RAM from the boot device (the reserved2 space is NOT loaded)

Note: The firmware does not load into the memory any data in the reserved2 space. Only the part of the load image that can continue loading the rest of the boot image is actually brought into system RAM by the

firmware. This allows the PowerPC Reference Platform boot partition to grow to any arbitrary size. It is important to allow the boot partition size to be larger than the system RAM because the size of entire boot partition could exceed available system RAM, especially in an entry-level system.

5.4 Transferring System Control to Load Image

After the load image has been loaded, the firmware transfers control to the entry point of the loader code. The state of the machine at this point is defined in Section 5.4.1, “System State.”

5.4.1 System State

When the firmware passes control to the software loaded through the boot mechanism, the following must be true:

/ a) For the area of System Memory where the load image resides, addressing must have been set up as physical address equals virtual address. This area must be covered by BAT registers.

/ b) IP (Interrupt prefix) bit in MSR (Machine Status Register) must be 0 to vector the interrupt to the System Memory.

c) System I/O address space must be in contiguous I/O mode. (For a discussion of the contiguous I/O mode used in the Reference Implementation, refer to Section 6.1, “Memory and I/O Map.”)

d) The video mode, if a graphics device is used, must be set to a bitmap mode with minimum resolution of 640x480x8.

e) The residual data must be available. Section 5.6.1, “Map of Residual Data Structure,” shows the data that the firmware must collect for an operating system. The memory addresses of the residual data and load image are passed on GPR3 and GPR4, respectively. Open Firmware must store the the address of the Open Firmware client interface at GPR5. For conventional firmware, the GPR5 must be set to 0.

f) Configuration and status information that are operating system independent must be stored in NVRAM. The NVRAM structure is shown in Section 5.5.5, “Map of NVRAM Data Structure.”

| g) It is strongly recommended that conventional firmware leave the system in Big-Endian mode. However, conventional firmware may leave the system in Little-Endian mode for client programs that the conventional firmware recognizes as expecting Little-Endian. On the other hand, Open Firmware must establish the system hardware Endian mode to be the same as the client program to be loaded.

/ h) Conventional firmware must disable the external interrupts. Open Firmware must enable the external interrupts.

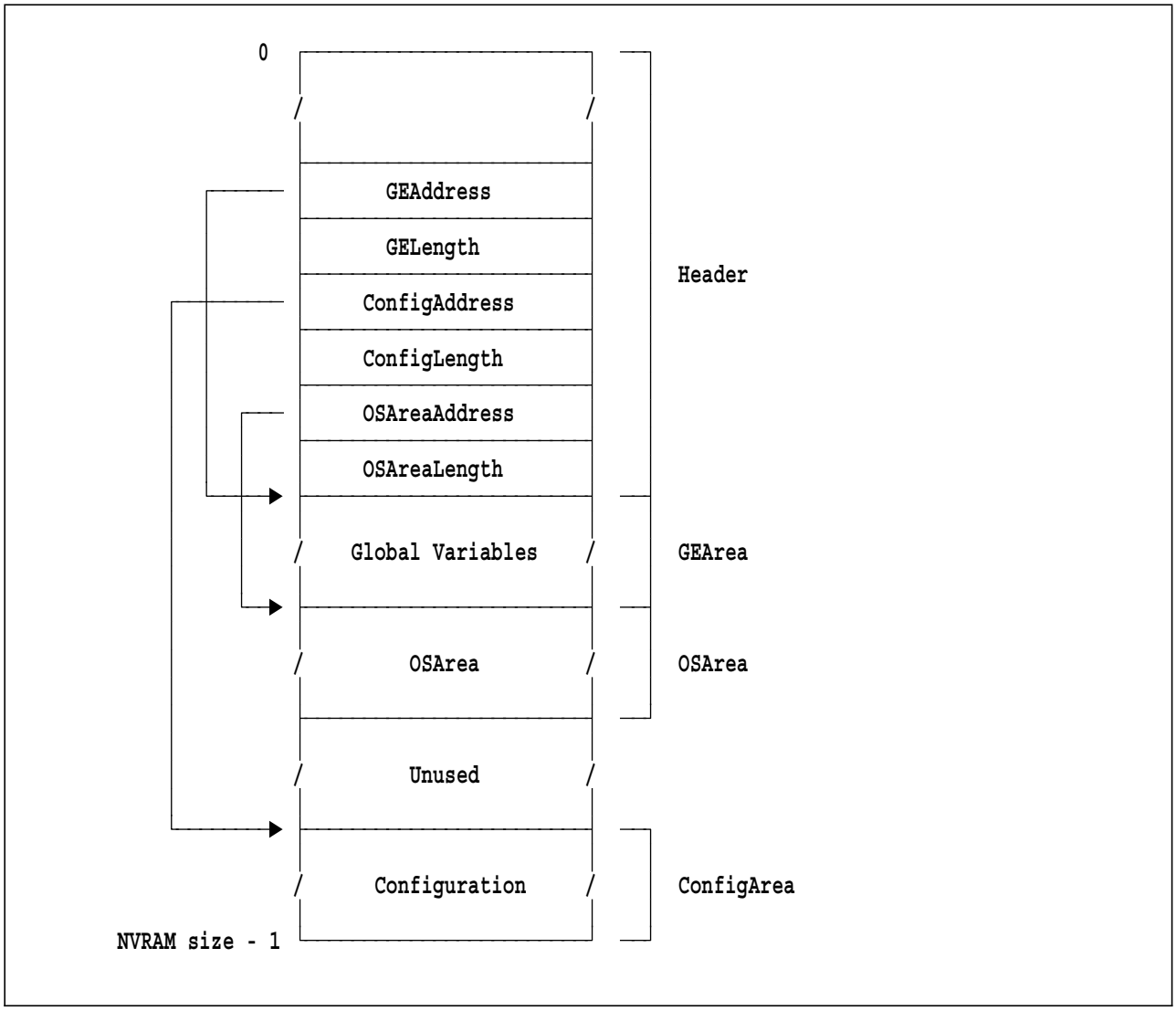
/ 5.4.2 Call-Back to Firmware

/ Call-back refers to a service request by software to the system firmware. In general, call-backs to firmware / are enabled through call-back entry points provided by the firmware.

/ Conventional firmware must not export any entry point for call-back to firmware. Open Firmware, on the / other hand, may allow call-back through its client interface. However, the call-back service must be limited / to support of the boot operations or to functions for which the firmware is efficient. This makes the run- / time operating system independent of the implementation of firmware. For the same reason, operating / systems must not export the call-back entry point for application programs.

5.5 NVRAM

/ NVRAM stores the system configuration data for the use of the firmware and the operating system. As
/ shown in Figure 17, NVRAM has four sections -- HEADER, GEArea, OSArea and ConfigArea. The
/ detailed structure of NVRAM is described in Section 5.5.5, “Map of NVRAM Data Structure.”



/ Figure 17. NVRAM Map

5.5.1 HEADER

/ The HEADER section starts at address 0. This section includes system variables such as version number of
/ NVRAM map, size of NVRAM, security fields, error logs, the start address and length of the other three
/ sections, etc. The version number in the HEADER can be used to identify updated versions of NVRAM
/ structures to accommodate future requirements.

/ The size, the version, the revision fields and the fields of security section in the header are maintained only
/ by the firmware. Operating systems may read these fields but must not modify any of them.

/ There are two error logging slots allocated in the header. The overrun bit may be used to preserve the oldest
/ error in a burst of error events.

5.5.2 Global Environment Area

The GEArea has definitions of global environment variables. The size and start address of the GEArea are specified by GEAddress and GELength in the HEADER section. The global environment variables must be settable from the firmware and operating systems. In general, the environment variable definition is a null-terminated string. The format is as shown below:

Name=<value>

Global variables are used mainly by firmware for cached data or by operating systems to communicate with firmware. These Global variables are architected. Architected environment variables defined to date are listed below:

Name	Comment
ClientIPAddr+	IP address of the machine
ServerIPAddr+	IP address of the BOOT server
GatewayIPAddr+	IP address of the Gateway
NetMask+	Network mask
boot-file++	Name of the file to be loaded by firmware
boot-device+++	Selected boot device
boot-path+++	Ordered list of the boot devices that the firmware searches during boot

\+ The values of these variables are represented in "dotted decimal notation" of the 32-bit IP address.

\+ + The value of *boot-file* is represented by the full path name of the file, starting from the root directory. Nodes in the path are separated by a "/."

\+ + + + The values of *boot-path* and *boot-device* are represented in the text representation of the path names as defined in the Open Firmware specification. The leaf node in the path name must have a device argument and the device argument must indicate the type of device. The possible device types include HARDDISK, NETWORK, CDROM and FLOPPY. For example, boot-device="/PCI/SCSI@2/HARDDISK@1,0:" refers to the hard disk that is configured as SCSI ID 1 on the SCSI controller that is configured as device number 2 on the PCI bus. Also, the boot path may list multiple boot devices, separated by a semicolon.

Open Firmware may use the GEArea for its configuration memory. When the GEArea is used, the names of the configuration variables must be the same as those defined in the Open Firmware specification and the script must be specified with "Script=<value>."

An operating system may use the GEArea for its own environment variables as non-architected variables. A naming convention is used to avoid potential conflict among the names of non-architected environment variables. The name of the non-architected global environment variables must be prefixed with a symbol of the operating system that created the variable. Following are the prefixes for the operating systems:

Operating system name	Prefix
AIX	"AIX-"
Windows NT	"WINNT-"
MK OS/2*	"MKOS2-"
Taligent	"TALIGENT-"
Solaris**	"SOLARIS-"
MK AIX	"MKAIX-"
MK	"MK-"

5.5.3 Operating System-Specific Area

The OSArea section is allocated for operating system-specific data. The size and the start address of the OSArea section are specified by OSAreaAddress and OSAreaLength in the HEADER section.

The OSArea space must be preserved between boots such that when the firmware transfers the system control to the operating system loader, the image in the OSArea must be same as when the system was last powered down.

While the global environment variables are preserved, the OSArea is a transient space and is not preserved when a different operating system is loaded. That is, if the LastOS does not match the ID of the operating system to be loaded, the OSArea space may be reconfigured by the operating system for its use. This allows the sharing of the OSArea among operating systems.

5.5.4 Configuration Area

The ConfigArea stores the configuration data for non-native ISA devices. The size and start address of the ConfigArea are specified by ConfigAddress and ConfigLength. This field is located at the tail of the NVRAM and grows toward lower addresses.

An operating system may store the configuration data of the devices. The data must be represented in the format of the compressed Plug and Play configuration packet.

The firmware must present this configuration data to the operating system on the next boot using residual data. Open Firmware must build a device tree with nodes for the devices that have configuration data in ConfigArea.

5.5.5 Map of NVRAM Data Structure

This section describes the structure of the NVRAM. The fields in the NVRAM structure must follow Big-Endian byte ordering. What follows is the current NVRAM header file, followed by an explanatory table that better defines the data structure.

5.5.5.1 NVRAM Header

The information in this subsection is the NVRAM header file that defines the contents of NVRAM.

```
/* Structure map for NVRAM on PowerPC Reference Platform */

/* All fields are either character/byte strings which are valid either
endian or they are big-endian numbers.

There are a number of Date and Time fields which are in RTC format,
big-endian. These are stored in UT (GMT).

For enum's: if given in hex then they are bit significant, i.e. only
one bit is on for each enum.
*/

#ifndef _NVRAM_
#define _NVRAM_

#define NVSIZE 4096       /* size of NVRAM */
#define OSAREASIZE 512    /* size of OSArea space */
#define CONFSIZE 1024     /* guess at size of Configuration space */
```

```
  typedef struct _SECURITY {
    unsigned long BootErrCnt;         /* Count of boot password errors */
    unsigned long ConfigErrCnt;       /* Count of config password errors */
    unsigned long BootErrorDT[2];     /* Date&Time from RTC of last error in pw */
    unsigned long ConfigErrorDT[2];   /* Date&Time from RTC of last error in pw */
    unsigned long BootCorrectDT[2];   /* Date&Time from RTC of last correct pw */
    unsigned long ConfigCorrectDT[2]; /* Date&Time from RTC of last correct pw */
    unsigned long BootSetDT[2];       /* Date&Time from RTC of last set of pw */
    unsigned long ConfigSetDT[2];     /* Date&Time from RTC of last set of pw */
    unsigned char Serial[16];         /* Box serial number */
    } SECURITY;

  typedef enum _OS_ID {
    Unknown = 0,
    Firmware = 1,
    AIX = 2,
    NT = 3,
|   MKOS2 = 4,
|   MKAIX = 5,
    Taligent = 6,
    Solaris = 7,
|   MK = 12
    } OS_ID;

  typedef struct _ERROR_LOG {
    unsigned char ErrorLogEntry[40]; /* To be architected */
    } ERROR_LOG;

| typedef enum _BOOT_STATUS {
|     BootStarted = 0x01,
|     BootFinished = 0x02,
|     RestartStarted = 0x04,
|     RestartFinished = 0x08,
|     PowerFailStarted = 0x10,
|     PowerFailFinished = 0x20,
|     ProcessorReady = 0x40,
|     ProcessorRunning = 0x80,
|     ProcessorStart = 0x0100
|     } BOOT_STATUS;

  typedef struct _RESTART_BLOCK {
      unsigned short Version;
      unsigned short Revision;
      unsigned long ResumeReserve1[2];
      volatile unsigned long BootStatus;
      unsigned long CheckSum; /* Checksum of RESTART_BLOCK */
|     void * RestartAddress;
|     void * SaveAreaAddr;
|     unsigned long SaveAreaLength;
    } RESTART_BLOCK;

  typedef enum _OSAREA_USAGE {
    Empty = 0,
    Used = 1
    } OSAREA_USAGE;

  typedef enum _PM_MODE {
    Suspend = 0x80,     /* Part of state is in memory */
    Normal = 0x00       /* No power management in effect */
```

```
  } PMMode;

typedef struct _HEADER {
  unsigned short Size;    /* NVRAM size in K(1024) */
  unsigned char Version;  /* Structure map different */
  unsigned char Revision; /* Structure map the same -
                             may be new values in old fields
                             in other words old code still works */
  unsigned short Crc1;    /* check sum from beginning of nvram to OSArea */
  unsigned short Crc2;    /* check sum of config */
  unsigned char LastOS;   /* OS_ID */
  unsigned char Endian;   /* B if big endian, L if little endian */
  unsigned char OSAreaUsage; /* OSAREA_USAGE */
  unsigned char PMMode;   /* Shutdown mode */
  RESTART_BLOCK RestartBlock;
  SECURITY Security;
  ERROR_LOG ErrorLog[2];

/* Global Environment information */
  void * GEAddress;
  unsigned long GELength;
  /* Date&Time from RTC of last change to Global Environment */
  unsigned long GELastWriteDT[2];

/* Configuration information */
  void * ConfigAddress;
  unsigned long ConfigLength;
  /* Date&Time from RTC of last change to Configuration */
  unsigned long ConfigLastWriteDT[2];
  unsigned long ConfigCount; /* Count of entries in Configuration */

/* OS dependent temp area */
  void * OSAreaAddress;
  unsigned long OSAreaLength;
  /* Date&Time from RTC of last change to OSAreaArea */
  unsigned long OSAreaLastWriteDT[2];
  } HEADER;


/* Here is the whole map of the NVRAM */
typedef struct _NVRAM_MAP {
  HEADER Header;
  unsigned char GEArea[NVSIZE-CONFSIZE-OSAREASIZE-sizeof(HEADER)];
  unsigned char OSArea[OSAREASIZE];
  unsigned char ConfigArea[CONFSIZE];
  } NVRAM_MAP;

#endif  /* ndef _NVRAM_ */
```

5.5.5.2 NVRAM Description

The information in this subsection describes the various fields in the NVRAM header file.

Field name	Address (Offset)	size (bytes)	Comment
		HEADER	
size	0x0	2	Size of the NVRAM in Kbytes

```
/       Version       0x2          1   Version number of NVRAM structure
/      Revision       0x3          1   Revision number of NVRAM Structure
|         *CRC1       0x4          2   Check sum from beginning of NVRAM
|                                      to OSArea
|         *CRC2       0x6          2   Check sum of ConfigArea
/      **LastOS       0x8          1   Identification of the operating system
/                                      that has loaded at last boot time
/        Endian       0x9          1   System is set to Little Endian mode
/                                      after boot if it has  "L", otherwise
/                                      system remains in Big Endian mode
/   OSAreaUsage       0xA          1   OSArea usage flag
/                                      Definition to be determined

/        PMMode       0xB          1   System state for power management
/                                      0x80 - System was suspended
/                                      0x40 - System was hibernated
/                                      0x00 - Normal
|      /* Beginning of restart block description record */
/       Version       0xC          2   Restart block version
/      Revision       0xE          2   Restart block revision
/  ResumeReserve1     0x10         8   Reserved for future use
/    BootStatus       0x18         4   Definition not architected
/                                      For firmware use to track
/                                      resume state
/      CheckSum       0X1C         4   Checksum of RESTART BLOCK
|  RestartAddress     0x20         4   Real address for operating system for
|                                      resuming from suspend mode
|  SaveAreaAddress    0x24         4   Real Address of reserved space for
|                                      resume firmware
|  SaveAreaLength     0x28         4   Length of space reserved for resume
|                                      firmware


/     /* Beginning of security section */
/    BootErrCnt       0x2C         4   Count of incorrect boot passwords entered
/   ConfigErrCnt      0x30         4   Count of incorrect config passwords entered
/      BootErrDT      0x34         8   Date and time in RTC format when last
/                                      incorrect boot password was entered
/    ConfigErrDT      0x3C         8   Date and time in RTC format when last
/                                      incorrect configuration password was
/                                      entered
/     BootLastDT      0x44         8   Date and time in RTC format when last
/                                      correct boot password was entered
/   ConfigLastDT      0x4C         8   Date and time in RTC format when last
/                                      correct configuration password was
/                                      entered
/      BootSetDT      0x54         8   Date and time in RTC format when last
/                                      boot password was set
/    ConfigSetDT      0x5C         8   Date and time in RTC format when last
/                                      configuration password was set
/         Serial      0x64        16   Box serial number

/     /* Beginning of the first error log record */
/  ErrorLogEntry      0x74        40   The error log record is still
/                                      being defined
/     /* Beginning of the second error log record */
/  ErrorLogEntry      0x9C        40

/     /* Pointers of the global environment area */
```

```
/     GEAddress      0xC4          4   Address offset of the global environment
/                                      variable area from the beginning of NVRAM
/      GELength      0xC8          4   Size of the global environment variable
/                                      area in bytes
/  GELastWriteDT     0xCC          8   Date and time in RTC format of last
/                                      modification to the global
/                                      environment variable area

/     /* Pointers of the configuration area */
/  ConfigAddress     0xD4          4   Address offset of the configuration area
/                                      from the beginning of NVRAM
/   ConfigLength     0xD8          4   Size of the configuration area in bytes
/   CLastWriteDT     0xDC          8   Date and time in RTC format of last
/                                      modification to the configuration
/                                      area
/    ConfigCount     0xE4          4   Count of entries in Configuration are

/     /* Pointers of the operating system specific area */
/  OSAreaAddress     0xE8          4   Address offset of the operating system
/                                      specific area from the beginning of NVRAM
/   OSAreaLength     0xEC          4   Size of the operating system specific
/                                      area in bytes
/  OSLastWriteDT     0xF0          8   Date and time in RTC format of last
/                                      modification to the operating
/                                      system specific area

/  ------------------------------------------------------------------------------
/                      Global Environment Variable Area
/  ------------------------------------------------------------------------------
/         GEArea ***(0xC4)  ***(0xC8)   Space for global environment variables

/  ------------------------------------------------------------------------------
/                      Configuration Area
/  ------------------------------------------------------------------------------
/         OSArea ***(0xE8)  ***(0xEC)   Space for operating system specific data

/  ------------------------------------------------------------------------------
/                      Operating System Specific Area
/  ------------------------------------------------------------------------------
/     ConfigArea ***(0xD4)  ***(0xD8)   Space for non-native device configuration
/                                       data
```

/ **Legend:**

```
|  *   16-bit CRC is computed using CCITT polynomial,
|      (x**16 + x**12 + x**5 + 1).
|      The new value of each CRC bit c(i) is computed as follows:
|          c(0)  = p(8)  XOR pd(0) XOR pd(4)                                  */
|          c(1)  = p(9)  XOR pd(1) XOR pd(5)                                  */
|          c(2)  = p(10) XOR pd(2) XOR pd(6)                                  */
|          c(3)  = p(11) XOR pd(0) XOR pd(3) XOR pd(7)                        */
|          c(4)  = p(12) XOR pd(1)                                            */
|          c(5)  = p(13) XOR pd(2)                                            */
|          c(6)  = p(14) XOR pd(3)                                            */
|          c(7)  = p(15) XOR pd(0) XOR pd(4)                                  */
|          c(8)  = pd(0) XOR pd(1) XOR pd(5)                                  */
|          c(9)  = pd(1) XOR pd(2) XOR pd(6)                                  */
|          c(10) = pd(2) XOR pd(3) XOR pd(7)                                  */
|          c(11) = pd(3)                                                      */
|          c(12) = pd(0) XOR pd(4)                                            */
```

```
|          c(13) = pd(1) XOR pd(5)                                          */
|          c(14) = pd(2) XOR pd(6)                                          */
|          c(15) = pd(3) XOR pd(7)                                          */

|       Where                                                               */
|          XOR      denotes bit wise exclusive or operation
|          p(i)     denotes bit i in the previous 16-bit CRC
|          d(i)     denotes bit i in the new 8-bit data
|          pd(i)    denotes p(i) XOR d(i)                                   */
```

/ ** These fields are specified with enumerated identifications of
/ operating systems. The identifications of operating systems are
/ defined as follows:

Operating system name	Identification
Unknown	0
Firmware	1
AIX	2
Windows NT	3
MKOS2	4
MKAIX	5
Taligent	6
Solaris	7
MK	12

/ *** Parentheses are used for indirect addressing. For example,
/ (0xF8) refers to the value in NVRAM at offset 0xF8 from the
/ beginning of NVRAM.

5.6 Residual Data

/ Residual data is used by conventional firmware to pass the system data collected by the firmware. The
| memory address of the residual data is passed on GPR3. The map of residual data is shown in Section
| 5.6.1, “Map of Residual Data Structure.” The terminology used in the residual data structure is defined in
| Section 5.6.2, “Plug and Play Configuration Structures.” This structure allows vendor-specific extensions.
| Some of those extensions that are being used for booting AIX are defined in Appendix J, “Plug and Play
| Extensions.” A dump of the residual data created on a Reference Implementation is shown in Appendix K,
| “Dump of Residual Data.”

/ **Note:** Open Firmware will provide the residual data as defined in Section 5.6.1, “Map of Residual Data
/ Structure,” to the operating systems. Future changes to the existing residual data may not be supported by
/ Open Firmware. It is strongly recommended that operating systems use the Open Firmware client interface
/ to collect the data necessary for the operating system instead of using residual data.

/ To avoid ambiguity in address alignment for objects in residual.h, the sizes of the data types used in Section
/ 5.6.1, “Map of Residual Data Structure,” are defined as follows:

Data type	Size in bits
char	8
short	16
long	32

Vital Product Data (VPD) must be supplied with the system. The boot process must know how to get this data and must place it in the residual data structure. For security purposes, the operating system must protect VPD in RAM from being modified. The list of the VPD is described in the residual structure.

One possible place to save the VPD is the area in the System ROM reserved for VPD. If VPD is stored in the same ROM as the boot code, care must be taken not to destroy or invalidate this information during boot ROM maintenance actions (e.g. replacing EPROM, or rewriting a Flash ROM).

5.6.1 Map of Residual Data Structure

```
/*------------------------------------------------------------------------*/
/*      Residual Data header definitions and prototypes                   */
/*------------------------------------------------------------------------*/

/* Structure map for RESIDUAL on PowerPC Reference Platform               */
/* residual.h - Residual data structure passed in r3.                     */
/*              Load point passed in r4 to boot image.                    */
/* For enum's: if given in hex then they are bit significant, i.e.        */
/* only one bit is on for each enum                                       */

#ifndef _RESIDUAL_
#define _RESIDUAL_

#define MAX_CPUS 16
#define MAX_MEMS 64
#define MAX_DEVICES 256
#define AVE_PNP_SIZE 32
#define MAX_MEM_SEGS 64

/*------------------------------------------------------------------------*/
/*               Public structures...                                     */
/*------------------------------------------------------------------------*/

  typedef enum _CACHE_TYPE {
|   NoneCAC = 0,
|   SplitCAC = 1,
|   CombinedCAC = 2
    } CACHE_TYPE;

  typedef enum _TLB_TYPE {
|   NoneTLB = 0,
|   SplitTLB = 1,
|   CombinedTLB = 2
    } TLB_TYPE;

  typedef enum _FIRMWARE_SUPPORT {
    Conventional = 0x01,
    OpenFirmware = 0x02,
    Diagnostics = 0x04,
    LowDebug = 0x08,
    Multiboot = 0x10,
    LowClient = 0x20,
    Hex41 = 0x40,
    FAT = 0x80,
    ISO9660 = 0x0100,
    } FIRMWARE_SUPPORT;

  typedef struct _VPD {

    /* Box dependent stuff */
    unsigned char PrintableModel[32];           /* Null terminated              */
|                                               /* Must be of the form:                  */
|                                /*       Manufacturer,0x0,Model,0x0,Serial,0x0,...      */
```

```
|    unsigned char Serial[64];                 /* A unique identifier for this box */
     unsigned short SpecVersion;                /* PPC Ref Pltfm version and revision     */
     unsigned short SpecRevision;               /*  on this machine                       */
     unsigned long FirmwareSupports;            /* See FirmwareSupport enum               */
     unsigned long NvramSize;                   /* Size of nvram in bytes -               */
                                 /*       neg if NVRAM reformatted because it was bad      */
     unsigned long NumSIMMSlots;
     unsigned long NumISASlots;
     unsigned long NumPCISlots;
     unsigned long NumPCMCIASlots;
     unsigned long NumMCASlots;
     unsigned long NumEISASlots;
|    unsigned long ProcessorHz;                  /* August 15, 1994 (MHz-->Hz)             */
|    unsigned long ProcessorBusHz;               /* August 15, 1994 (MHz-->Hz)             */
|    unsigned long PCIHz;                        /* August 15, 1994 (MHz-->Hz)             */
|    unsigned long TimeBaseDivisor;              /* (Bus clocks per timebase tic) * 1000   */

     /* Derivable from CpuType but included for convenience */
     unsigned long WordWidth;                   /* Word width in bits 601 - 32            */
     unsigned long PageSize;                    /* Page size in bytes 601 - 4k            */
     unsigned long CoherenceBlockSize;          /* In bytes 601 - 32                      */
     unsigned long GranuleSize;                 /* In bytes 601 - 32                      */

     /* Cache and TLB variables */
     unsigned long CacheSize;                   /* Cache size in Kbytes 601 - 32k         */
     CACHE_TYPE CacheAttrib;                    /* Combined ,split or None                */
     unsigned long CacheAssoc;                  /* Associativity 601 - 8                  */
     unsigned long CacheLineSize;               /* Cache line size 601 - 64; 2 sectors    */
                                                /*    per line                            */
     unsigned long I_CacheSize;                 /* 601 - 32k                              */
     unsigned long I_CacheAssoc;                /* 601 - 8                                */
     unsigned long I_CacheLineSize;             /* 601 - 64; 2 sectors per line           */
     unsigned long D_CacheSize;                 /* 601 - 32k                              */
     unsigned long D_CacheAssoc;                /* 601 - 8                                */
     unsigned long D_CacheLineSize;             /* 601 - 64; 2 sectors per line           */
     unsigned long TLBSize;                     /* Number of TLBs on the system 601 - 256 */
     TLB_TYPE TLBAttrib;                        /* Combined I+D or split                  */
     unsigned long TLBAssoc;                    /* Associativity 601 - 2                  */
     unsigned long I_TLBSize;                   /* 601 - 256                              */
     unsigned long I_TLBAssoc;                  /* 601 - 2                                */
     unsigned long D_TLBSize;                   /* 601 - 256                              */
     unsigned long D_TLBAssoc;                  /* 601 - 2                                */

     void * ExtendedVPD;
     } VPD;

   typedef enum _DEVICE_FLAGS {
     Failed = 0x1000,                           /* 1 - device failed POST code tests      */
     Static = 0x0800,                           /* 0 - dynamically configurable;          */
                                                /* 1 - static                             */
     Dock = 0x0400,                             /* 0 - not a docking station device;      */
                                                /* 1 is a docking station device          */
     IPLable = 0x0200,                          /* 0 - not an IPLable device;             */
                                                /* 1 - IPLable                            */
     Configurable = 0x0100,                     /* 1 - device is configurable             */
     Disablable = 0x80,                         /* 1 - device can be disabled             */
     PowerManaged = 0x40,                       /* 0 - not managed; 1 - managed           */
     ReadOnly = 0x20,
     Removable = 0x10,
```

```
    ConsoleIn = 0x08,
    ConsoleOut = 0x04,
    Input = 0x02,
    Output = 0x01
    } DEVICE_FLAGS;

  typedef enum _BUS_ID {
    ISADEVICE = 0x01,
    EISADEVICE = 0x02,
    PCIDEVICE = 0x04,
    PCMCIADEVICE = 0x08,
    PNPISADEVICE = 0x10,
    MCADEVICE = 0x20,
|   PROCESSORDEVICE = 0x80                    /* devices on local processor bus */
|                                             /* August 15, 1994               */
    } BUS_ID;

  typedef struct _DEVICE_ID {
    unsigned long BusId;                      /* See BUS_ID enum above         */
    unsigned long DevId;
    unsigned long SerialNum;
    unsigned long Flags;                      /* See DEVICE_FLAGS enum above   */
    unsigned char BaseType;                   /* See pnp.h for bit definitions */
    unsigned char SubType;                    /* See pnp.h for bit definitions */
    unsigned char Interface;                  /* See pnp.h for bit definitions */
    unsigned char Spare;
    } DEVICE_ID;

  typedef union _BUS_ACCESS {
    struct _PnPAccess{
      unsigned char CSN;
      unsigned char LogicalDevNumber;
      unsigned short ReadDataPort;
      } PnPAccess;
    struct _ISAAccess{
      unsigned char SlotNumber;
      unsigned char LogicalDevNumber;
      unsigned short ISAReserved;
      } ISAAccess;
    struct _MCAAccess{
      unsigned char SlotNumber;
      unsigned char LogicalDevNumber;
      unsigned short MCAReserved;
      } MCAAccess;
    struct _PCMCIAAccess{
      unsigned char SlotNumber;
      unsigned char LogicalDevNumber;
      unsigned short PCMCIAReserved;
      } PCMCIAAccess;
    struct _EISAAccess{
      unsigned char SlotNumber;
      unsigned char FunctionNumber;
      unsigned short EISAReserved;
      } EISAAccess;
    struct _PCIAccess{
      unsigned char BusNumber;
      unsigned char DevFuncNumber;
      unsigned short PCIReserved;
      } PCIAccess;
```

```
|    struct _BridgeAccess{                  /* August 15, 1994             */
|      unsigned char BusNumber;             /* August 15, 1994             */
|      unsigned char NumberOfSlots;         /* number of slots in BusNumber */
|                                           /* August 15, 1994             */
|      unsigned short BridgeReserved;       /* August 15, 1994             */
|      } BridgeAccess;                      /* August 15, 1994             */
     } BUS_ACCESS;

   /* Per logical device information */
   typedef struct _PPC_DEVICE {
     DEVICE_ID DeviceId;
     BUS_ACCESS BusAccess;

     /* The following three are offsets into the DevicePnPHeap */
     /* All are in PnP compressed format                       */
     unsigned long AllocatedOffset;               /* Allocated resource description */
     unsigned long PossibleOffset;                /* Possible resource description  */
     unsigned long CompatibleOffset;              /* Compatible device identifiers  */
     } PPC_DEVICE;

   typedef struct _PPC_CPU {
     unsigned long CpuType;                       /* Result of mfpvr -            */
                                                  /*  might be different rev level */
     unsigned long PerCpuSerial;
     unsigned long L2_CacheSize;                  /* L2 Cache Information */
     unsigned long L2_CacheAsc;
     } PPC_CPU;

   typedef struct _PPC_MEM {
     unsigned long SIMMSize;                      /* 0 - absent or bad, 8M, 32M in K(1024)   */
     } PPC_MEM;

   typedef enum _MEM_USAGE {    /* See specification, section 6.1 -                          */
                                /*  Reference implementation memory map                      */
     ResumeBlock = 0x4000,                        /* for use by power management            */
     SystemROM = 0x2000,                          /* Flash memory (populated)               */
     UnPopSystemROM = 0x1000,                     /* Unpopulated part of SystemROM area     */
     IOMemory = 0x0800,                           /* 3G to 4G - 16M                         */
     SystemIO = 0x0400,                           /* 2G to 3G - next 4 are details within it */
     SystemRegs = 0x0200,                         /* 3G - 8M to 3G                          */
     PCIAddr = 0x0100,                            /* 2G + 16M to 3G - 8M Used for SCSI I/O   */
     PCIConfig = 0x80,                            /* 2G + 8M to 2G + 16M                    */
     ISAAddr = 0x40,                              /* 2G to 2G + 8M                          */
     Unpopulated = 0x20,                          /* Unpopulated part of System Memory      */
     Free = 0x10,                                 /* Free part of System Memory             */
     BootImage = 0x08,                            /* BootImage part of System Memory        */
     FirmwareCode = 0x04,                         /* FirmwareCode part of System Memory     */
     FirmwareHeap = 0x02,                         /* FirmwareHeap part of System Memory     */
     FirmwareStack = 0x01                         /* FirmwareStack part of System Memory    */
     } MEM_USAGE;

   typedef struct _MEM_MAP {
     unsigned long Usage;                         /* See MEM_USAGE above                    */
     unsigned long BasePage;                      /* i.e. page number measured in 4K pages   */
     unsigned long PageCount;
     } MEM_MAP;

   typedef struct _RESIDUAL {
     unsigned long ResidualLength;                /* Length of Residual                     */
```

```
    unsigned short Version;                     /* of this data structure               */
    unsigned short Revision;                    /* of this data structure               */

    VPD VitalProductData;

    unsigned long ActualNumCpus;
    PPC_CPU Cpus[MAX_CPUS];

    unsigned long TotalMemory;                  /* Total amount of memory installed     */
    unsigned long GoodMemory;                   /* Total amount of good memory          */
    unsigned long ActualNumMemSegs;
    MEM_MAP Segs[MAX_MEM_SEGS];
    unsigned long ActualNumMemories;
    PPC_MEM Memories[MAX_MEMS];

    unsigned long ActualNumDevices;
    PPC_DEVICE Devices[MAX_DEVICES];
    unsigned char DevicePnPHeap[2*MAX_DEVICES*AVE_PNP_SIZE];

    } RESIDUAL;

#endif  /* ndef _RESIDUAL_ */
```

5.6.2 Plug and Play Configuration Structures

The following describes Plug and Play terminology used in both the NVRAM and residual data structures.

```
/*------------------------------------------------------------------------*/
/*      Plug and Play header definitions                                  */
/*------------------------------------------------------------------------*/

/* Structure map for PnP on PowerPC Reference Platform                     */
/* See Plug and Play ISA Specification, Version 1.0, May 28, 1993.  It     */
/* (or later versions) is available on Compuserve in the PLUGPLAY area.    */
/* This code has extensions to that specification, namely new short and    */
/* long tag types for platform dependent information                       */

/* Warning: LE notation used throughout this file                          */

/* For enum's: if given in hex then they are bit significant, i.e.         */
/* only one bit is on for each enum                                        */

#ifndef _PNP_
#define _PNP_

#define MAX_MEM_REGISTERS 9
#define MAX_IO_PORTS 20
#define MAX_IRQS 7
#define MAX_DMA_CHANNELS 7

/* Device Base Type Codes */

typedef enum _PnP_BASE_TYPE {
  Reserved = 0,
  MassStorageDevice = 1,
  NetworkInterfaceController = 2,
  DisplayController = 3,
  MultimediaController = 4,
```

```
/    Memory = 5,
/    BridgeController = 6,
/    CommunicationsDevice = 7,
/    SystemPeripheral = 8,
/    InputDevice = 9
/    } PnP_BASE_TYPE;

/  /* Device Sub Type Codes */

/  typedef enum _PnP_SUB_TYPE {
/    SCSIController = 0,
/    IDEController = 1,
/    FloppyController = 2,
/    IPIController = 3,
/    OtherMassStorageController = 0x80,

/    EthernetController = 0,
/    TokenRingController = 1,
/    FDDIController = 2,
/    OtherNetworkController = 0x80,

/    VGAController= 0,
/    SVGAController= 1,
/    XGAController= 2,
/    OtherDisplayController = 0x80,

/    VideoController = 0,
/    AudioController = 1,
/    OtherMultimediaController = 0x80,

/    RAM = 0,
/    FLASH = 1,
/    OtherMemoryDevice = 0x80,

/    HostProcessorBridge = 0,
/    ISABridge = 1,
/    EISABridge = 2,
/    MicroChannelBridge = 3,
/    PCIBridge = 4,
/    PCMCIABridge = 5,
/    OtherBridgeDevice = 0x80,

/    RS232Device = 0,
/    ATCompatibleParallelPort = 1,
/    OtherCommunicationsDevice = 0x80,

/    ProgrammableInterruptController = 0,
/    DMAController = 1,
/    SystemTimer = 2,
/    RealTimeClock = 3,
|    L2Cache = 4,                        /* L2 Cache               August 15, 1994 */
|    NVRAM = 5,                          /* NVRAM                  August 15, 1994 */
|    PowerManagement = 6,                /* Power Management       August 15, 1994 */
|    CMOS = 7,                           /* CMOS                   August 15, 1994 */
/    OtherSystemPeripheral = 0x80,

/    KeyboardController = 0,
/    Digitizer = 1,
/    MouseController = 2,
```

```
/    OtherInputController = 0x80
/    } PnP_SUB_TYPE;

/  /* Device Interface Type Codes */

/  typedef enum _PnP_INTERFACE {
/    General = 0,
/    GeneralSCSI = 0,
/    GeneralIDE = 0,
/    ATACompatible = 1,
/    GeneralFloppy = 0,
/    Compatible765 = 1,
/    GeneralIPI = 0,

/    GeneralEther = 0,
/    GeneralToken = 0,
/    GeneralFDDI = 0,

/    GeneralVGA = 0,
/    GeneralSVGA = 0,
/    GeneralXGA = 0,

/    GeneralVideo = 0,
/    GeneralAudio = 0,

/    GeneralRAM = 0,
/    GeneralFLASH = 0,

/    GeneralHostBridge = 0,
/    GeneralISABridge = 0,
/    GeneralEISABridge = 0,
/    GeneralMCABridge = 0,
/    GeneralPCIBridge = 0,
|    PCIBridgeDirect = 0,                  /* August 15, 1994 */
|    PCIBridgeIndirect = 1,                /* August 15, 1994 */
/    GeneralPCMCIABridge = 0,

/    GeneralRS232 = 0,
/    COMx = 1,
/    Compatible16450 = 2,
/    Compatible16550 = 3,
/    GeneralParPort = 0,
/    LPTx = 1,

/    GeneralPIC = 0,
/    ISA_PIC = 1,
/    EISA_PIC = 2,
/    GeneralDMA = 0,
/    ISA_DMA = 1,
/    EISA_DMA = 2,
/    GeneralTimer = 0,
/    ISA_Timer = 1,
/    EISA_Timer = 2,
/    GeneralRTC = 0,
/    ISA_RTC = 1,
|    GeneralCMOS = 0,                      /* CMOS with 1 byte of address and data */
|                                          /* August 15, 1994                      */
|    InlineL2 = 0,                         /* inline L2 cache                      */
|                                          /* August 15, 1994                      */
```

```
|    LookasideL2 = 1,                    /* lookaside L2 cache                  */
|                                        /* August 15, 1994                     */
|    BufLookasideL2 = 2,                 /* buffer lookaside L2 cache           */
|                                        /* August 15, 1994                     */
|    GeneralNVRAM = 0,                   /* NVRAM with 2 bytes of address       */
|                                        /* August 15, 1994                     */
|                                        /* and 1 byte of data registers        */
|                                        /* August 15, 1994                     */
|    GeneralPowerManagement = 0          /* Power Management                    */
|                                        /* August 15, 1994
/    } PnP_INTERFACE;

/  typedef enum _CONSOLE_TYPE {
/    Console_NoConsole = 0,                    /* Console unknown        */
/    Console_Serial    = 1,                    /* Serial console         */
/    Console_Video     = 2,                    /* Video in 8 bpp graphics */
/    Console_Video16   = 3,                    /* Video in 5,6,5 graphics */
/    Console_Video24   = 6,                    /* Video in 8,8,8 graphics */
/    Console_VGA       = 7,                    /* Video in VGA text mode  */
/    } CONSOLE_TYPE;

/  typedef enum _DISKETTE_TYPE {               /* see BOC-AR-07012, p 9-5 Figure 9-2 */
/    D525x2M = 2,                            /* 5.25 high density                  */
/    D35x2M = 4,                             /* 3.5 high density                   */
/    D35x4M = 6
/    } DISKETTE_TYPE;

/  typedef enum _DISKETTE_FEATURE {
/    MediaSense = 0x01,
/    AutoEject = 0x02
/    } DISKETTE_FEATURE;

/  typedef enum _PnP_SUBTAG {
/    Extended = 0,
/                                           /* Small platform tags ...            */
/    KeyboardType = 1,                       /* Keyboard id                        */
/    KeyboardKeys = 2,                       /* Number of keys                     */
/    KeyboardCaps = 3,                       /* Key caps i.e. language             */
/    MouseButtons = 6,                       /* Number of buttons                  */
/    MouseLR = 7,                            /* 0 - right handed, 1 left handed    */
/    ModemParity = 10,                       /* E - even, O - odd, N - none        */
/    ModemSpeed = 11,                        /* 1200, 2400, 9600, 14400, etc.      */
/    ModemStop = 12,                         /* 1 or 2                             */
/    DisplayID = 20,                         /* 4 bits, e.g. 1010 = 8514, etc.     */
/    DisplayHor = 21,                        /* Horizontal resolution, e.g. 640    */
/    DisplayVer = 22,                        /* Vertical Resolution, e.g. 480      */
/    DisplayBuffer = 23,                     /* Address of display buffer          */
/    DisplayType = 24,                       /* See CONSOLE_TYPE enum              */
/    DisplayRow = 25,                        /* Number of Rows - VGA and Serial    */
/    DisplayCol = 26,                        /* Number of Columns - VGA and Serial */
/    DisketteType = 30,                      /* See DISKETTE_TYPE enum             */
/    DisketteFeature = 31,                   /* Diskette features - media sense,   */
/                                           /* auto eject                         */
/    SCSILun = 40,                           /* lun                                */
/    DiskType = 41,
/    DiskSize = 42,
/                                           /* Large platform tags ...            */
/    LargeIO = 1,                            /* 4 byte I/O addresses               */
/    MemoryIndirectAddr1 = 2,                /* For low-client indirect addresses  */
```

```
/    MemoryIndirectAddr2 = 3                      /* For low-client indirect addresses  */
/    } PnP_SUBTAG;

/  /* PnP resources */

/  /* Compressed ASCII is 5 bits per char; 00001=A ... 11010=Z */

/  typedef struct _SERIAL_ID {
/    unsigned char VendorID0; /* bit7=0; bit(6:2)=1st compressed ASCII; bit(1:0)  */
/                             /*  2nd compressed ASCII bit(4:3)                   */
/    unsigned char VendorID1; /* bit(7:5) 2nd compressed ASCII bit(2:0); bit(4:0) */
/                             /*  3rd compressed ASCII                            */
/    unsigned char VendorID2; /* Product number - vendor assigned          */
/    unsigned char VendorID3; /* Product number - vendor assigned          */

/  /* Serial number is to provide uniqueness if more than one board of same */
/  /* type is in system.  Must be "FFFFFFFF" if feature not supported.      */

/    unsigned char Serial0;   /* Unique serial number bits (7:0)   */
/    unsigned char Serial1;   /* Unique serial number bits (15:8)  */
/    unsigned char Serial2;   /* Unique serial number bits (23:16) */
/    unsigned char Serial3;   /* Unique serial number bits (31:24) */
/    unsigned char Checksum;
/    } SERIAL_ID;

/  typedef enum _PnPItemName {
/    Unused = 0,
/    PnPVersion = 1,
/    LogicalDevice = 2,
/    CompatibleDevice = 3,
/    IRQFormat = 4,
/    DMAFormat = 5,
/    StartDepFunc = 6,
/    EndDepFunc = 7,
/    IOPort = 8,
/    FixedIOPort = 9,
/    Res1 = 10,
/    Res2 = 11,
/    Res3 = 12,
/    SmallPlatformItem = 13,
/    SmallVendorItem = 14,
/    EndTag = 15,
/    MemoryRange = 1,
/    ANSIIdentifier = 2,
/    UnicodeIdentifier = 3,
/    LargeVendorItem = 4,
/    MemoryRange32 = 5,
/    MemoryRangeFixed32 = 6,
/    LargePlatformItem = 16
/    } PnPItemName;

/  /* Define a bunch of access functions for the bits in the tag field */

/  /* Tag type - 0 = small; 1 = large */
/  #define tag_type(t) (((t) & 0x80)>>7)
/  #define set_tag_type(t,v) (t = (t & 0x7f) | ((v)<<7))
```

```
/  /* Small item name is 4 bits - one of PnPItemName enum above */
/  #define tag_small_item_name(t) (((t) & 0x78)>>3)
/  #define set_tag_small_item_name(t,v) (t = (t & 0x07) | ((v)<<3))

/  /* Small item count is 3 bits - count of further bytes in packet */
/  #define tag_small_count(t) ((t) & 0x07)
/  #define set_tag_count(t,v) (t = (t & 0x78) | (v))

/  /* Large item name is 7 bits - one of PnPItemName enum above */
/  #define tag_large_item_name(t) ((t) & 0x7f)
/  #define set_tag_large_item_name(t,v) (t = (t | 0x80) | (v))

/  /* a PnP resource is a bunch of contiguous TAG packets ending with an end tag */

/  typedef union _PnP_TAG_PACKET {
/    struct _S1_Pack{                          /* VERSION PACKET    */
/      unsigned char Tag;                      /*  small tag = 0x0a */
/      unsigned char Version[2];               /*  PnP version, Vendor version */
/      } S1_Pack;

/    struct _S2_Pack{                          /* LOGICAL DEVICE ID PACKET          */
/      unsigned char Tag;                      /*  small tag = 0x15 or 0x16         */
/      unsigned char DevId[4];                 /*  Logical device id       */
/      unsigned char Flags[2];                 /*  bit(0) boot device;     */
/                                              /*  bit(7:1) command in range x31-x37 */
/                                              /*  bit(7:0) command in range x28-x3f */
/                                              /*   (optional)                       */
/      } S2_Pack;

/    struct _S3_Pack{                          /* COMPATIBLE DEVICE ID PACKET     */
/      unsigned char Tag;                      /*  small tag = 0x1c               */
/      unsigned char CompatId[4];              /*  Compatible device id */
/      } S3_Pack;

/    struct _S4_Pack{                          /* IRQ PACKET                                    */
/      unsigned char Tag;                      /*  small tag = 0x22 or 0x23                     */
/      unsigned char IRQMask[2];              /*  bit(0) is IRQ0, ..; bit(0) is IRQ8 .. */
/      unsigned char IRQInfo;                  /*  optional; assume bit(0)=1; else              */
/                                              /*     bit(0) - high true edge sensitive         */
/                                              /*     bit(1) - low true edge sensitive          */
/                                              /*     bit(2) - high true level sensitive        */
/                                              /*     bit(3) - low true level sensitive         */
/                                              /*     bit(7:4) - must be 0                      */
/      } S4_Pack;

/    struct _S5_Pack{                          /* DMA PACKET              */
/      unsigned char Tag;                      /*  small tag = 0x2a       */
/      unsigned char DMAMask;                  /*  bit(0) is channel 0 ... */
/      unsigned char DMAInfo;
/      } S5_Pack;

/    struct _S6_Pack{                          /* START DEPENDENT FUNCTION PACKET       */
/      unsigned char Tag;                      /*  small tag = 0x30 or 0x31             */
/      unsigned char Priority;                 /*  Optional; if missing then x01; else  */
/                                              /*                  x00 = best possible */
/                                              /*                  x01 = acceptible    */
/                                              /*                  x02 = sub-optimal,  */
/                                              /*     but functional                   */
/      } S6_Pack;
```

```
/    struct _S7_Pack{                          /* END DEPENDENT FUNCTION PACKET */
/      unsigned char Tag;                      /*  small tag = 0x38            */
/      } S7_Pack;

/    struct _S8_Pack{                          /* VARIABLE I/O PORT PACKET       */
/      unsigned char Tag;                      /*  small tag x47                 */
/      unsigned char IOInfo;                   /*  x0 = decode only bits(9:0);   */
/                                              /*  x01 = decode bits(15:0)       */
/      unsigned char RangeMin[2];              /*  Min base address     */
/      unsigned char RangeMax[2];              /*  Max base address     */
/      unsigned char IOAlign;                  /*  base alignment, increment in  */
/                                              /*   1 byte blocks                */
/      unsigned char IONum;                    /*  number of contiguous I/O ports */
/      } S8_Pack;

/    struct _S9_Pack{                          /* FIXED I/O PORT PACKET          */
/      unsigned char Tag;                      /*  small tag = 0x4b              */
/      unsigned char Range[2];                 /*  base address 10 bits */
/      unsigned char IONum;                    /*  number of contiguous I/O ports */
/      } S9_Pack;

/    struct _S13_Pack{
/      unsigned char Tag;                      /* small tag = 0x6m m = 8-F   */
/      unsigned char SubTag;                   /* device dependent tag       */
/      unsigned char Data[6];                  /* Platform defined */
/      } S13_Pack;

/    struct _S14_Pack{                         /* VENDOR DEFINED PACKET      */
/      unsigned char Tag;                      /*  small tag = 0x7m m = 1-7  */
|      unsigned char Type;                     /*  00=non-IBM                */
/      unsigned char Data[6];                  /*  Vendor defined  */
/      } S14_Pack;

/    struct _S15_Pack{                         /* END PACKET                  */
/      unsigned char Tag;                      /*  small tag = 0x78 or 0x79   */
/      unsigned char Check;                    /*  optional - checksum        */
/      } S15_Pack;

/    struct _L1_Pack{                          /* MEMORY RANGE PACKET         */
/      unsigned char Tag;                      /*  large tag = 0x81           */
/      unsigned char Count0;                   /*  x09                        */
/      unsigned char Count1;                   /*  x00
/      unsigned char Data[9];                  /*  a variable array of bytes, */
/                                              /*    count in tag                    */
/      } L1_Pack;

/    struct _L2_Pack{                          /* ANSI ID STRING PACKET               */
/      unsigned char Tag;                      /*  large tag = 0x82                   */
/      unsigned char Count0;                   /*  Length of string                   */
/      unsigned char Count1;
/      unsigned char Identifier[1];            /*  a variable array of bytes, */
/                                              /*    count in tag                    */
/      } L2_Pack;

/    struct _L3_Pack{                          /* UNICODE ID STRING PACKET            */
/      unsigned char Tag;                      /*  large tag = 0x83                   */
/      unsigned char Count0;                   /*  Length + 2 of string               */
/      unsigned char Count1;
```

```
/     unsigned char Country0;                  /*  TBD                                  */
/     unsigned char Country1;                  /*  TBD                                  */
/     unsigned char Identifier[1];             /*  a variable array of bytes, */
/     count in tag
/     } L3_Pack;

/   struct _L4_Pack{                           /* VENDOR DEFINED PACKET                 */
/     unsigned char Tag;                       /*  large tag = 0x84                     */
/     unsigned char Count0;                    /*                                       */
/     unsigned char Count1;                    /*                                       */
|     unsigned char Type;                      /*  00=non-IBM                           */
/     unsigned char Data[1];                   /*  a variable array of bytes, */
/                                              /*    count in tag                       */
/     } L4_Pack;

/   struct _L5_Pack{
/     unsigned char Tag;                       /*  large tag = 0x85                     */
/     unsigned char Count0;                    /*  Count = 17                           */
/     unsigned char Count1;
/     unsigned char Data[17];
/     } L5_Pack;

/   struct _L6_Pack{
/     unsigned char Tag;                       /*  large tag = 0x86                      */
/     unsigned char Count0;                    /*  Count = 9                             */
/     unsigned char Count1;
/     unsigned char Data[9];
/     } L6_Pack;

/   struct _L16_Pack{
/     unsigned char Tag;                       /*  large tag = 0x90                      */
/     unsigned char SubTag;
/     unsigned char Count0;
/     unsigned char Count1;
/     union _L16_Data{
/       unsigned char Data[1];                 /*  a variable array of bytes,     */
/                                              /*      count in tag                       */
/       struct _L16_IO{                        /*  SubTag = 1                             */
/         unsigned char RangeMin[4];           /*  Min base address              */
/         unsigned char RangeMax[4];           /*  Max base address              */
/         unsigned char IOAlign;               /*  base alignment, inc in 1 byte blocks     */
/         unsigned char IONum[4];              /*  number of contiguous I/O ports */
/         } L16_IO;
/       } L16_Data;
/     } L16_Pack;

/   } PnP_TAG_PACKET;

/ #endif  /* ndef _PNP_ */
```

5.7 Open Firmware Extension for PowerPC Reference Platform

/ Open Firmware uses a hardware-independent and extendable interpretive language, FCode. FCode has a
/ Forth dictionary that is extended for boot. Using FCode, the Open Firmware process builds a device tree,
/ which is a hierarchical data structure describing system hardware. Open Firmware also uses configuration
/ memory, with which a user can affect the behavior of Open Firmware functions.

/ This section describes the specific requirements and recommendations of Open Firmware for the PowerPC
/ Reference Platform.

/ 5.7.1 Open Firmware Requirements

/ In order to be PowerPC Reference Platform compliant, Open Firmware must provide the following:

/ a) the Open Firmware-compliant client interface
/ b) the Open Firmware-compliant device interface
/ c) Bi-Endian booting capability

/ **Note:** Bi-Endian booting establishes the Endian mode of the hardware system to be the same as that of the
/ operating system loaded. Furthermore, Bi-Endian booting includes the capability of handling call-backs
/ from either Big-Endian or Little-Endian operating systems.

/ 5.7.2 Open Firmware Process

/ The main purpose of using Open Firmware on a PowerPC Reference Platform system is to support a boot
/ process which is independent of system configuration. Open Firmware achieves this system-independent
/ boot process by allowing processor-independent boot drivers to reside on adaptor ROMs and by providing
/ methods to use these drivers. These drivers are coded in FCode.

/ For plug-in devices that may not have these FCode ROMs, it is strongly recommended that the system
/ provide an alternate method of making Open Firmware compatible drivers available. One implementation
/ approach would be for these driver images in FCode to be installed from media (e.g. a diskette) to Flash
/ ROM or some other non-volatile storage. The system firmware would need to provide a utility that would
/ perform this installation. Once installed, the FCode driver image would be used by Open Firmware in the
/ same way as the driver in ROM on a plug-in device. The FCode driver image that is loaded via this mech-
/ anism must follow the Open Firmware bus specification for the bus to which the device attaches. It is
/ recommended that a minimum of 32 KB of non-volatile storage be allocated for these drivers.

/ To provide boot services, Open Firmware requires four basic elements:

/ • Forth programming language
/ • A Forth dictionary
/ • A device tree
/ • Configuration memory

/ Before the Open Firmware process starts, the FCode interpreter must be initialized. A PowerPC Open
/ Firmware implementation must perform the following steps during the boot process:

/ a) Initialize built-in devices: Device nodes and drivers for built-in devices reside in Open Firmware and are
/ permanently installed in the device tree. Some amount of testing may be performed as part of this step
/ to insure that the device is functioning correctly.

/ b) Configure the non-Plug and Play ISA devices stored in NVRAM: NVRAM may have configuration
/ information for non-PNP ISA devices in ConfigArea. Open Firmware must include the nodes in the
/ device tree for those non-PnP ISA devices.

/ c) Probe plug-in devices: The plug-in devices are located, their device nodes are added on the device tree,
/ and the nodes are set with proper property values and associated with their methods. If the device is a
/ boot device, its device driver must be available at the end of this process.

/ d) Switch Endian mode: Firmware must establish the Endian mode of the hardware system to be con-
/ sistent with that of the operating system to be loaded.

6.0 Reference Implementation

This section describes a reference implementation of a PowerPC Reference Platform-compliant system. This Reference Implementation is intended as an example of one way to build to the PowerPC Reference Platform architecture. Information in this section should not be construed as specifying the only implementation possible or recommended.

If more detailed information on the Reference Implementation is needed, two design kits are available from IBM Microelectronics Division. One design kit shows this implementation using a PowerPC 601 processor; the second design kit shows this implementation using a PowerPC 603 processor. These design kits contain additional descriptive information, schematic diagrams showing connections of components, and sample hardware.

The recommended PowerPC Reference Platform system design for a desktop as shown in Figure 18 contains a processor complex, an I/O complex, and various types of devices and adaptors. Within this figure, a dashed line is used to show optional devices and connections. For instance, the graphics subsystem may be attached directly to the PCI bus or attached via a PCI connector.

The processor complex consists of a 601 processor, a memory controller and PCI bridge, System Memory, and an open slot for a second level of cache (e.g. L2 Cache) or a processor upgrade to a higher-speed 601 or 604 processor chip. The Processor and I/O complexes are connected via the PCI bridge and PCI bus. Designs based on the current system could have a processor running at 50, 66, 80, or 100 MHz with corresponding processor bus speeds of 50, 66, 40 or 50 MHz. The System Memory could have up to eight memory slots and, depending upon the choice of memory SIMMs, may have memory ranging from 8 to 256 MB using the 8- or 32-MB SIMMs. The design does not preclude using SIMMs of other sizes (e.g. 4, 16, 64 MB) as they become available in compatible packaging. Memory is parity checking.

The I/O complex consists of the PCI bus, various connections, and an I/O controller. One or more PCI connectors for PCI adaptor cards may be connected to the PCI bus. A SCSI subsystem controller is connected to the PCI bus. Flash ROM is located on the PCI bus. The Flash ROM, or any updatable initialization program storage media (Flash ROM is implementation dependent and technology may present a faster and less costly solution) is used to bring up the system. The graphics subsystem may be connected directly to the PCI bus or optionally may be plugged into a PCI connector. Optionally, a bus bridge to other tertiary buses may be provided.

The I/O controller is a bridge to the ISA bus and ISA adaptors that supply the remainder of the I/O ports. Non-volatile RAM (NVRAM) is located with the non-volatile Real-Time Clock and battery on the ISA bus. The NVRAM is used for recording error and configuration information that will be retained when a system is powered down.

The following subsections describe the components used to build this Reference Implementation. No recommendation is made of these specific components; they are shown as one approach to implement this example of a PowerPC Reference Platform-compliant system. The subsystems are presented for the processor complex, the I/O complex, and attached peripheral devices and interfaces. Following this information are some basic configuration alternatives and more detail on the upgrade slot.

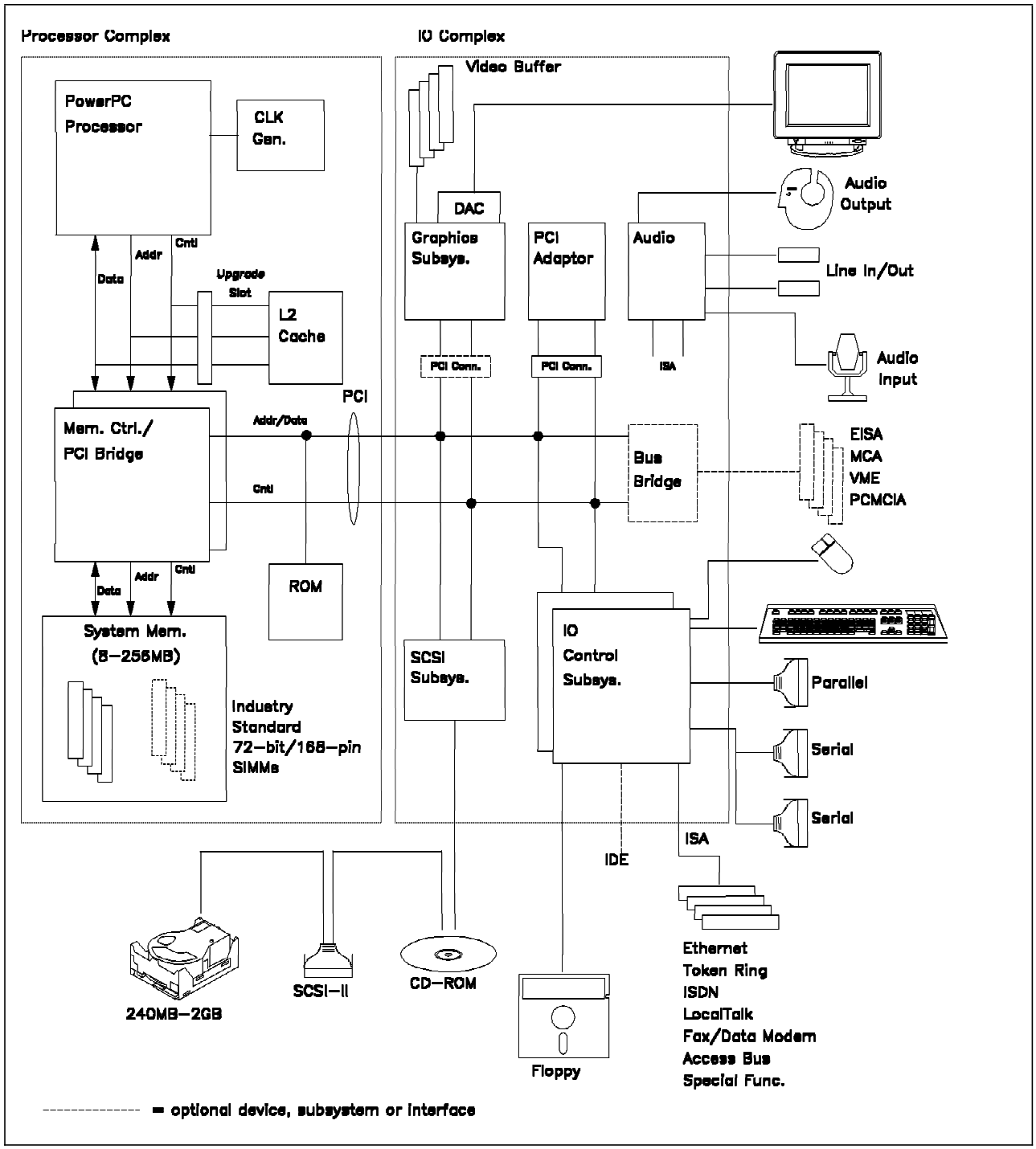


Figure 18. PowerPC Reference Platform Recommended Desktop System

6.1 Memory and I/O Map

The memory and I/O map for a PowerPC Reference Platform-compliant system is shown in Figure 19. The left side of this figure shows the view of memory from the PowerPC processor. The right side of this figure shows the view of memory of the I/O master doing I/O addressing or memory addressing. As shown on the left side of the figure, the address space is split into three areas: a System Memory portion with addresses from 0 to 2 GB, a System I/O portion which stretches from 2 GB to 3 GB, and an I/O Memory area which covers 3 GB to 4 GB.

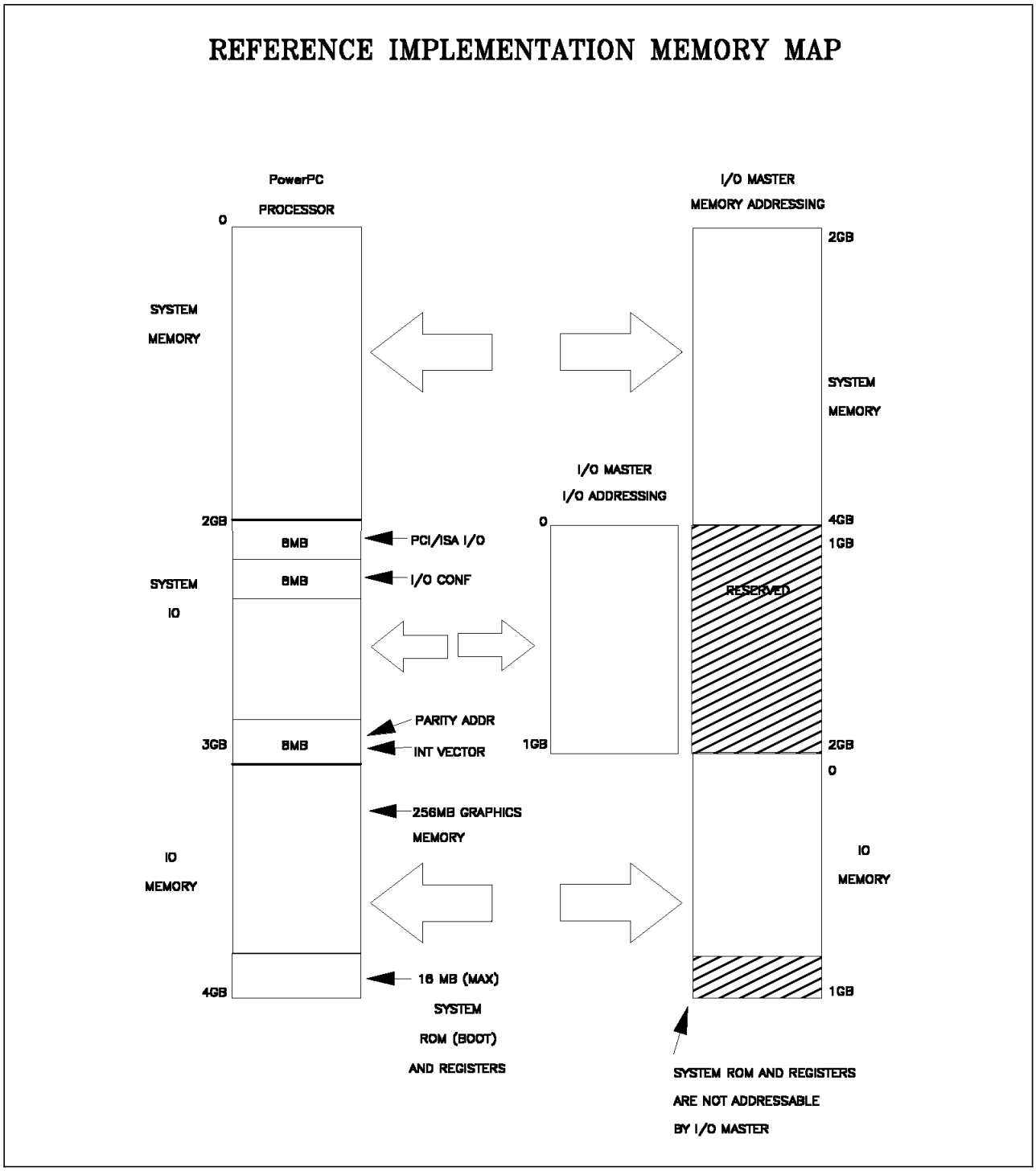


Figure 19. Reference Implementation Memory Map

The System Memory is addressed by the operating system and user applications executing in the PowerPC processor. Access to System Memory by the PowerPC processor is in the 0 to 2 GB range. System Memory accessed by a I/O master will be addressed in the 2 to 4 GB range, as shown on the right side of the figure.

The System I/O address space is used for input and output to the devices attached to the secondary or tertiary I/O buses (e.g. PCI and ISA). The first 8 MB of the System I/O address space are used for most system I/O. This address space includes interfaces to devices such as system registers, timers, parallel and

serial ports and NVRAM. For a detailed list of which devices are included in this area and where the devices appear within this area, refer to Section 6.1.5, “I/O Device Mapping.” The next 8 MB are used for the I/O configuration. Refer to Section 6.1.7, “I/O Configuration Space Mapping,” for mappings of the I/O configuration information. The first page after this 8 MB of configuration information is used for SCSI I/O as an implementation option. At the bottom of the System I/O address space is an 8-MB area which is reserved. Within this area is a page for the memory parity address and a second page for the interrupt vector. Refer to Section 6.1.8, “Additional System I/O Mappings,” for mappings of this area.

The I/O Memory area is used for graphics memory, System ROM, and other I/O Memory-based devices. The bottom 16 MB of the I/O Memory area is reserved for fixed address ISA devices and adaptors. The potential for video memory with a 256-MB memory space is shown in the figure. The size of this space will have to be traded off against other I/O Memory use in different configurations. The System ROM and space for system registers is allocated to the top 16 MB of the I/O Memory area address space. System ROM is implemented using Flash ROM. The System ROM contains the code for power-on self tests (POST), code for establishing the initial configuration of the system, code for booting, and system-specific information.

As shown on the right side of the figure, the I/O master has two modes of addressing. The I/O master doing memory address mode transfers will see System Memory as having addresses from 2 GB to 4 GB. The I/O Memory is addressed in the range of 0 to 1 GB. This maintains compatibility with fixed-address ISA devices and adaptors. The upper 16 MB of this space is reserved for the System ROM and system registers and is not addressable by the I/O master. In the memory addressing mode, the PCI master cannot address the space from 1 GB to 2 GB. The I/O master doing I/O addressing mode transfers will see the System I/O memory space as having addresses from 0 to 1 GB.

6.1.1 Processor View of the Memory Map

Figure 20 shows the view of memory from the PowerPC processor. Processor addresses in the 0 through 2 GB - 1 range access System Memory. These memory cycles will not be passed to the I/O bus. Addresses in the range 2 GB through 2 GB + 64 KB - 1 are for ISA-standard I/O. Addresses from 2 GB + 64 KB through 2 GB + 8 MB - 1 are reserved and used in an alternate ISA-standard I/O addressing mode (see below). Addresses between 2 GB + 8 MB through 3 GB - 1 are used by the software for I/O. Within this range, the space 2 GB + 8 MB through 2 GB + 16 MB - 1 is used for I/O configuration and 3 GB - 8 MB through 3 GB - 1 is used for the memory parity address register and the interrupt vector register. Processor memory cycles in the 2 GB through 3 GB - 1 range will be run by the memory controller as an I/O cycle with the most significant bit of the address set to zero. This translation puts the address in the 0 through 1 GB - 1 range for the I/O addresses. Processor memory cycles in the 3 GB through 4 GB - 16 MB - 1 range are used to address I/O Memory. The memory controller will run these addresses as I/O Memory cycles with the two most significant bits set to zero. As an implementation, only the first 16 MB of this space is forwarded to the ISA memory space. Addresses in the range 4 GB - 16 MB through 4 GB - 1 are used for the system ROM and system registers. The memory controller will access the ROM using I/O cycles.

This version of the memory map is called the *contiguous memory map* because the 64 KB of ISA I/O is contained in 64 contiguous KBs. There is no protection mechanism implemented for this area other than the page protection inherent in any memory space. Some operating systems such as Windows NT will use this I/O addressing scheme. Device drivers performing I/O in this area risk interfering with each other. Within the kernel, this risk is limited through testing. For device drivers contained within applications, the risk will be avoided by avoiding applications having device drivers which may corrupt other device drivers.

Figure 21 shows an alternate approach for handling the ISA I/O addresses. This approach is called the *discontiguous memory map*. The only difference between this map and the contiguous memory map is within the 8 MB address space beginning at 2 GB. Within this space, the objective is to have a single device on a memory page. This Reference Implementation translates this 64 KB of I/O addressing to allow for

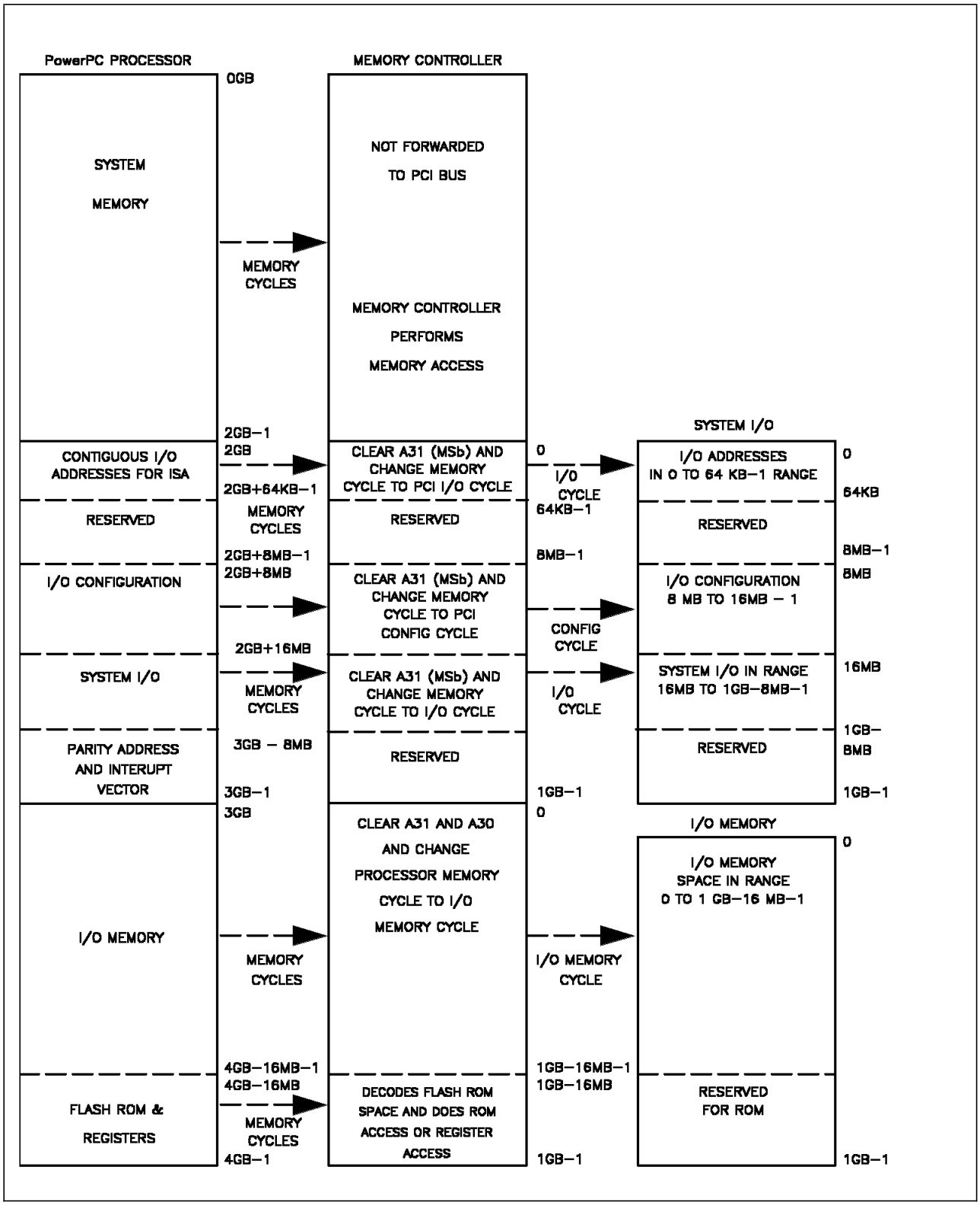


Figure 20. Reference Implementation Contiguous Memory Map

2048 devices, each having 32 bytes. In this address space, each device has the first 32 bytes of a 4096-byte page.

When in the discontiguous mode, the memory controller will translate the address and issue a PCI I/O cycle to that address. For addresses in the range of 2 GB through 2 GB + 8 MB - 1, the memory controller will

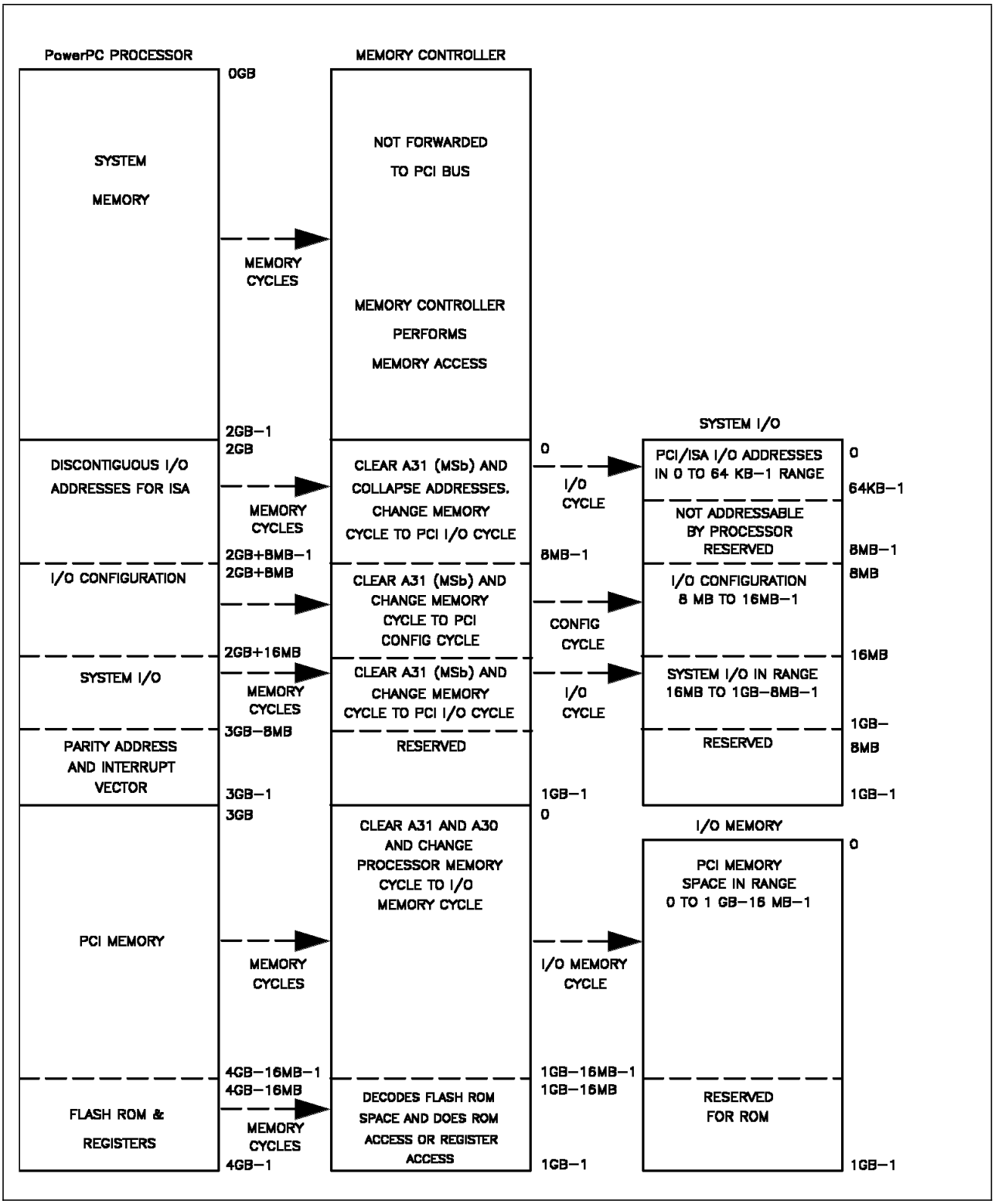


Figure 21. Reference Implementation Discontiguous Memory Map

clear the most significant bit of the address and will convert it to the 0 through 64 KB - 1 range. Then the memory controller will issue an I/O cycle to this converted address. This address translation by the memory controller makes the low-order 5 bits of the I/O address equivalent to the low-order 5 bits of the processor I/O address. The next 7 bits of the processor address are ignored. The processor address bits 9 through 19 are placed in the next-higher-order 11 bits of the I/O address. The remaining processor high-order address

bits indicate that this address is in the ISA address space. They must be zero, except the most significant bit, which is set to one to indicate System I/O or I/O Memory. **Notice that the processor numbers bits with zero as the most significant bit (MSb)**. Table 13 shows this address translation using a 16-bit ISA address which by convention has the LSb labeled 0.

Table 13. Translation of ISA I/O Addresses

Processor Address		ISA I/O Address
31 LSb	is assigned to	0 LSb
30	is assigned to	1
29	is assigned to	2
28	is assigned to	3
27	is assigned to	4
26 - 20	ignore	
19	is assigned to	5
18	is assigned to	6
17	is assigned to	7
16	is assigned to	8
15	is assigned to	9
14	is assigned to	10
13	is assigned to	11
12	is assigned to	12
11	is assigned to	13
10	is assigned to	14
9	is assigned to	15 MSb
8 - 1	must be 0	
0 MSb	must be 1	

The switch between the contiguous memory map and the discontiguous memory map will be controlled by software setting a configuration register. The design of the system should not preclude switching modes as required for the software environment. For instance, an operating system which is designed to use the discontiguous mode may switch to the contiguous mode when emulating other operating environments for applications which have drivers using the contiguous mode.

With the device mapping which is shown in the column for the discontiguous memory map in Table 14, most devices are contained within a page. Using memory page protection, device drivers can be protected from each other. Operating systems such as OS/2 built on the Workplace environment will use this discontiguous memory map.

6.1.2 I/O Master View of the I/O Map

Figure 22 shows an I/O master view of the memory map when in the I/O mode. The memory controller does not perform any operations for I/O cycles generated by I/O masters or the I/O bus bridge.

6.1.3 I/O Master View of the Memory Map

Figure 23 shows an I/O master view of the memory map when the I/O master is performing memory transfers. The memory controller ignores any memory operation for I/O master memory cycles which fall in the 16 MB through 2 GB - 1 range of I/O Memory. The memory controller performs a System Memory access for I/O master memory cycles which fall into the 2 GB through 4 GB - 1 range of I/O Memory. For accesses in this range, the memory controller complements the most significant bit of the memory address from the I/O master. This modified address, which is in the range 0 through 2 GB - 1, is used for System Memory access and cache snoop.

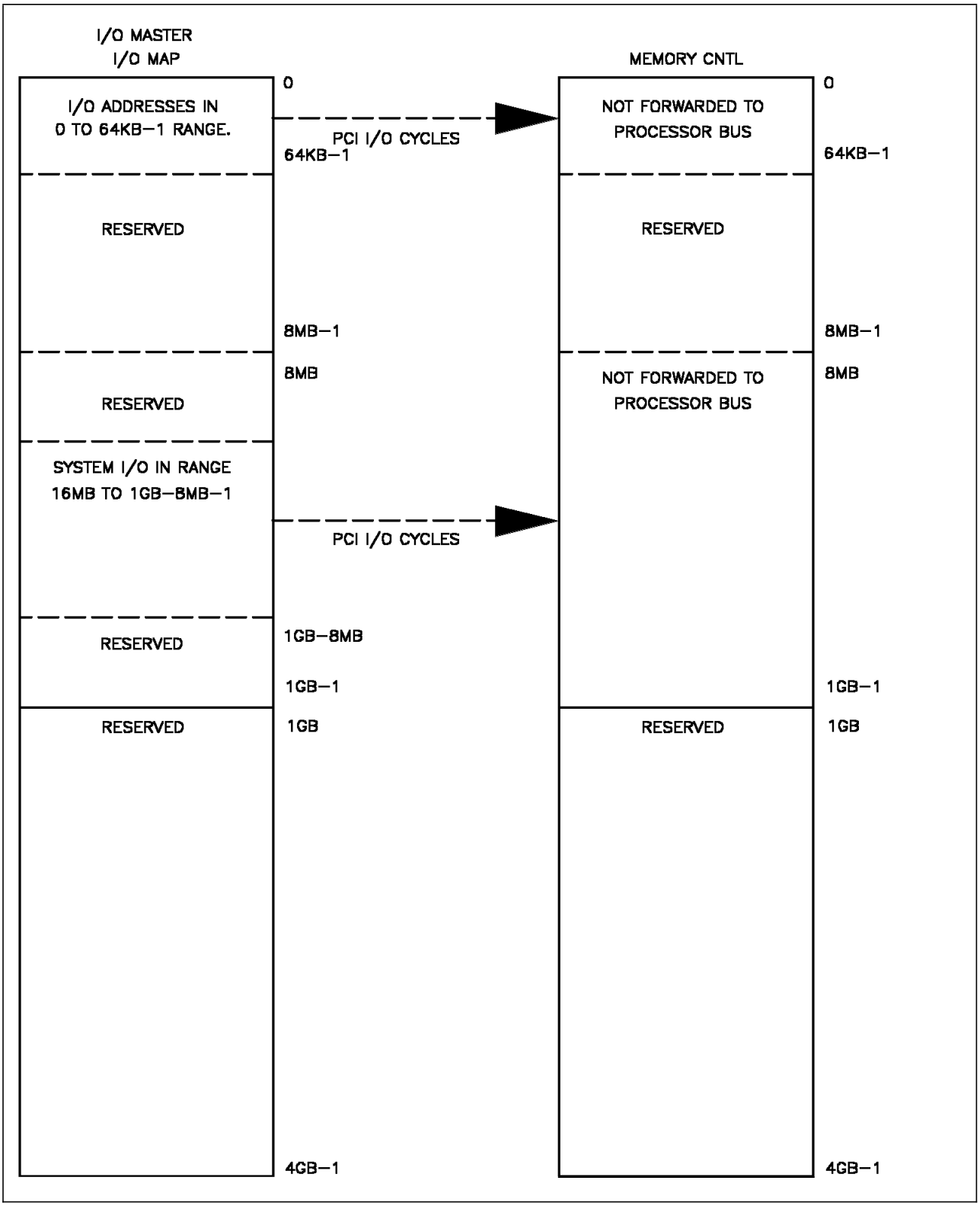


Figure 22. I/O Master View of I/O Map

/ In addition to these memory transfers, ISA masters may store to System Memory in the range of 0 to 16 MB. Within this 16-MB range, holes are defined by ISA usage conventions. I/O Memory cycles on behalf of ISA masters in the range 0 to 16 MB will be forwarded to System Memory in the range of 0 to 16 MB.
/ If a system has an ISA bus, then it is recommended that PCI devices not use I/O Memory in the range of 0
/ to 16 MB, and that PCI devices in the I/O Memory range of 2 GB to 2 GB + 16 MB avoid conflicts with

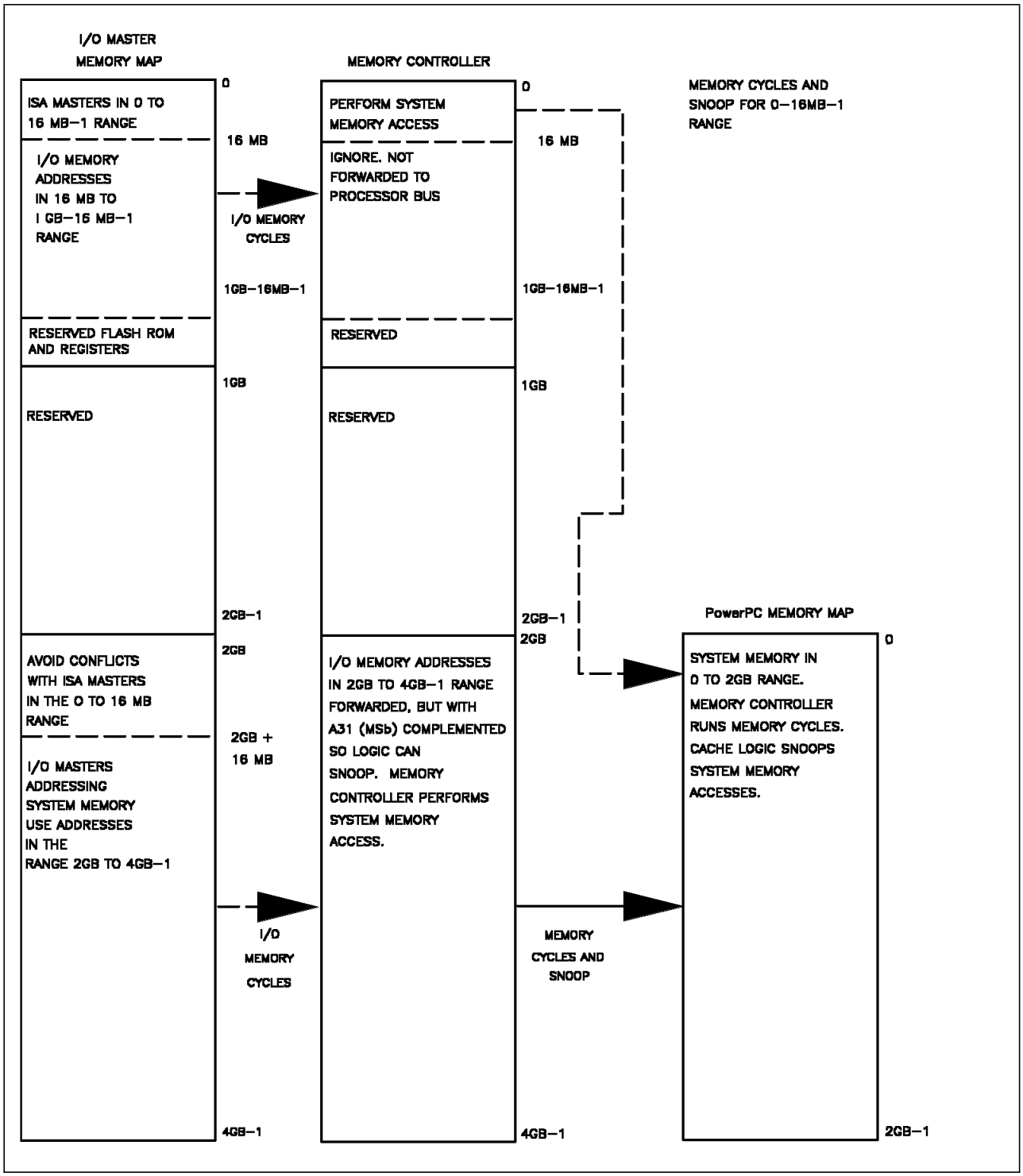


Figure 23. I/O Master View of Memory Map

/ the ISA devices in the System Memory range of 0 to 16 MB. In addition, low System Memory space reserved for interrupt vectors and used by some operating systems must be avoided by both PCI and ISA masters.

6.1.4 Rationale for Memory Model

The reason this design was chosen was to avoid a hole in the System Memory space and to allow the System Memory space and I/O space to expand without interference in each other's areas. In the current PC/AT-type memory map, I/O Memory located on I/O devices resides at an address between X'A0000' through X'DFFFF'. Without the memory remapping shown in this design, there would be a hole in System Memory. This hole would be used for the I/O devices. In addition, a second storage area for video adaptors and System ROM would have to be assigned either at the top of System Memory or as a continuation of the I/O area. If I/O space was required to be increased beyond the current 64 KB identified by the Intel specification, then the hole in System Memory would grow and software would have to change to accommodate this change. If video and ROM storage were part of this area, they would contribute to the likelihood of this area growing. If video and ROM storage were above the System Memory and the System Memory grew because more than 32 bits of addressing are available, software would have to change to accommodate this growth. The decision was made to use the most significant bit in the address to distinguish I/O space from System Memory space. The hardware makes the translation based on this bit. Supporting this decision is the information which tells the start and length of each area. This information makes the areas relocatable and provides flexibility for future growth in the address space.

6.1.5 I/O Device Mapping

Table 14 gives the ISA addresses, corresponding Reference Implementation system addresses for both the / contiguous I/O map and the discontiguous I/O map, and the functions at each address. Refer to device and / chip manufacturer literature for documentation of the utilization of these ISA addresses. The contiguous I/O map has the 64 KB of I/O space mapped into a contiguous 64 KB of space starting at 2 GB. The discontiguous I/O map has this 64 KB of I/O space mapped into the first 8 MB above 2 GB. These two I/O maps are described in Section 6.1, "Memory and I/O Map." Within Table 14, several I/O address ranges are marked as "Do not use" for the discontiguous I/O map. These spaces would be in the same 32 bytes as another device, and if both devices were used, the protection that is enforced by the mapping of one device per page would not be possible.

Table 14 (Page 1 of 4). System I/O Address Map

Address in Hex			
	PowerPC Processor Map		
ISA Space	Contiguous	Discontiguous	Function
0000-000F	8000 0000-000F	8000 0000-000F	DMA 1 Registers and Control
0020-0021	8000 0020-0021	8000 1000-1001	Interrupt 1 Control and Mask
0040-0043	8000 0040-0043	8000 2000-2003	Timer 1
0060	8000 0060	8000 3000	Reset UBus IRQ 12 and Keyboard Chip Select
0061	8000 0061	8000 3001	NMI Status and Control
/ 0062	8000 0062	8000 3002	Keyboard Reserved
0064	8000 0064	8000 3004	Keyboard Chip Select
/ 0066-0067	8000 0066-0067	8000 3006-3007	Keyboard Reserved
0070	8000 0070	8000 3010	RTC Address and NMI Enable
0071	8000 0071	8000 3011	RTC Read/Write
0074	8000 0074	8000 3014	NVRAM Address STB 0

Table 14 (Page 2 of 4). System I/O Address Map

	Address in Hex			
		PowerPC Processor Map		
	ISA Space	Contiguous	Discontiguous	Function
	0075	8000 0075	8000 3015	NVRAM Address STB 1
	0076	8000 0076	8000 3016	Reserved for NVRAM
	0077	8000 0077	8000 3017	NVRAM Data Port
	0078-007C	8000 0078-007C	8000 3018-301C	Reserved
	0080-008F	8000 0080-008F	8000 4000-400F	DMA Page Registers 0-7
	0090	8000 0090	8000 4010	DMA Page Register Reserved
	0092	8000 0092	8000 4012	Port 92 Register (Used for LE mode and Soft Reset)
	0094-0096	8000 0094-0096	8000 4014-4016	DMA Page Register Reserved
	0098	8000 0098	8000 4018	DMA Page Register Reserved
	009C-009E	8000 009C-009E	8000 401C-401E	DMA Page Register Reserved
	009F	8000 009F	8000 401F	DMA Low Page Register Refresh
	00A0-00A1	00A0-00A1	8000 5000-5001	Interrupt 2 Control and Mask
	00C0-00CF	8000 00C0-00CF	8000 6000-600F	DMA 2 Address Registers
	00D0-00DF	8000 00D0-00DF	8000 6010-601F	DMA 2 Control Registers
	00F0	8000 00F0	8000 7010	Coprocessor Error Register Reserved
	0170-0177	8000 0170-0177	8000 B010-B017	Secondary Disk IDE
	01F0-01F7	8000 01F0-01F7	8000 F010-F017	Primary Disk IDE
/	0220-0227	8000 0220-0227	8001 1000-1007*	Serial Port 3 (Secondary)
/	0228-022F	8000 0228-022F	8001 1008-100F*	Serial Port 4 (Secondary)
	0238-023F	8000 0238-023F	8001 1018-101F	Serial Port 4
/	0278-027A	8000 0278-027A	8001 3018-301A	Parallel Port 3
/	027A-027F	8000 027A-027F	8001 301A-301F	Reserved Parallel Port 3
/	02E0-02E7	8000 02E0-02E7	8001 7000-7007*	Serial Port 4 (Tertiary)
/	02E8-02EF	8000 02E8-02EF	8001 7008-700F*	Serial Port 3 (Tertiary) or 4 (fourth choice)
	02F8-02FF	8000 02F8-02FF	8001 7018-701F	Serial Port 2
	0338-033F	8000 0338-033F	8001 9018-901F	Serial Port 3
	0370-0371	8000 0370-0371	8001 B010-B011	Diskette Drive Control (Secondary) Reserved
	0372	8000 0372	8001 B012	Diskette Drive Control (Secondary)
	0373-0377	8000 0373-0377	8001 B013-B017	Diskette Drive Control (Secondary) Reserved and Secondary Disk IDE (376-7)
/	0378-037A	8000 0378-037A	Do not use	Parallel Port 2
/	037B-037F	8000 037B-037F	Do not use	Reserved Parallel Port 2

Table 14 (Page 3 of 4). System I/O Address Map

Address in Hex			
	PowerPC Processor Map		
ISA Space	Contiguous	Discontiguous	Function
0398	8000 0398	8001 C018	Super I/O Index Address
0399	8000 0399	8001 C019	Super I/O Data Address
/ 03BC-03BE	8000 03BC-03BE	8001 D01C-D01E	Parallel Port 1
/ 03BF	8000 03BF	8001 D01F	Reserved Parallel Port 1
03E0-03E3	8000 03E0-03E3	8001 F000-F003	PCMCIA Carrier Card Setup
/ 03E8-03EF	8000 03E8-03EF	Do not use	Serial Port 3 (fourth choice)
03F0-03F1	8000 03F0-03F1	Do not use	Diskette Drive Control (Prime) Reserved
03F2	8000 03F2	Do not use	Diskette Drive Control (Prime)
03F3-03F7	8000 03F3-03F7	Do not use	Diskette Drive Control (Prime) Reserved and Primary Disk IDE (3F6-7)
03F8-03FF	8000 03F8-03FF	8001 F018-F01F	Serial Port 1
040B	8000 040B	8002 000B	DMA 1 Extended Mode Register
0410-041F	8000 0410-041F	8002 0010-001F	DMA Scatter/Gather Command/Status
0420-042F	8000 0420-042F	8002 1000-100F	DMA Scatter/Gather Descriptor (Ch 0-3)
0434-043F	8000 0434-043F	8002 1014-101F	DMA Scatter/Gather Descriptor (Ch 5-7)
0481-0483	8000 0481-0483	8002 4001-4003	DMA High Page Registers
0487	8000 0487	8002 4007	DMA High Page Registers
0489	8000 0489	8002 4009	DMA High Page Registers
048A-048B	8000 048A-048B	8002 400A-400B	DMA High Page Registers
04D6	8000 04D6	8002 6016	DMA 2 Extended Mode Register
0800-0802	8000 0800-0802	8004 0000-0002	Reserved
0803	8000 0803	8004 0003	SIMM ID (32/8 MB)
0804	8000 0804	8004 0004	SIMM Presence
0805-0807	8000 0805-0807	8004 0005-0007	Reserved
0808	8000 0808	8004 0008	Hardfile Light Register
0809-080B	8000 0809-080B	8004 0009-000B	Reserved
080C	8000 080C	8004 000C	Equipment Present
080D-080F	8000 080D-080F	8004 000D-000F	Reserved
0810	8000 0810	8004 0010	Password Protect 1 Register
0811	8000 0811	8004 0011	Reserved
0812	8000 0812	8004 0012	Password Protect 2 Register
0813	8000 0813	8004 0013	Reserved
0814	8000 0814	8004 0014	L2 Invalidate

Table 14 (Page 4 of 4). System I/O Address Map

Address in Hex			
	PowerPC Processor Map		
ISA Space	Contiguous	Discontiguous	Function
0815-0817	8000 0815-0817	8004 0015-0017	Reserved
0818	8000 0818	8004 0018	Key Lock Position Register
0819-081B	8000 0819-081B	8004 0019-001B	Reserved
081C	8000 081C	8004 001C	System Control
081D-081F	8000 081D-081F	8004 001D-001F	Reserved
0820	8000 0820	8004 1000	Memory Controller Size Programming Register
0821	8000 0821	8004 1001	Memory Controller Timing Programming Register
0830	8000 0830	8004 1010	Audio Index Register
0831	8000 0831	8004 1011	Audio Indexed Data Register
0832	8000 0832	8004 1012	Audio Status Register
0833	8000 0833	8004 1013	Audio PIO Data Register
0840	8000 0840	8004 2000	Read Memory Parity Error
0842	8000 0842	8004 2002	Read Processor DPE Error
0843	8000 0843	8004 2003	Clear Processor DPE Error
0844	8000 0844	8004 2004	Read Illegal Transfer Error
0850	8000 0850	8004 2010	ISA I/O Map Type
/ 0852	8000 0852	8004 2012	System Board Identification
1378-137D	8000 1378-137D	8009 B018-B01D	Parallel Port 4
15E8-15EA	8000 15E8-15EA	800A F008-F00A	Reserved
4100-4101	8000 4100-4101	8020 8000-8001	Reserved

/ **Legend:**

/ ***** Do not use if conflicts with another used serial port

Software Development Note: The definition of some of these registers, particularly registers that are marked as reserved or those defined in the X'8xx' range, may change for other reference implementations. The address and content should be referenced abstractly by software systems.

Some system I/O addresses which are defined for this implementation are described in the following subsections. **For the registers defined in this subsection, the bit significance of these registers is given in Big-Endian (BE) form (the processor's viewpoint).** The naming convention used in the schematics and the vendor data books is Little-Endian (LE) Mode. Figure 24 shows the naming convention used in these subsections.

Within this section some bits are marked as reserved. Indeterminate data will be in these bits when the byte is read.

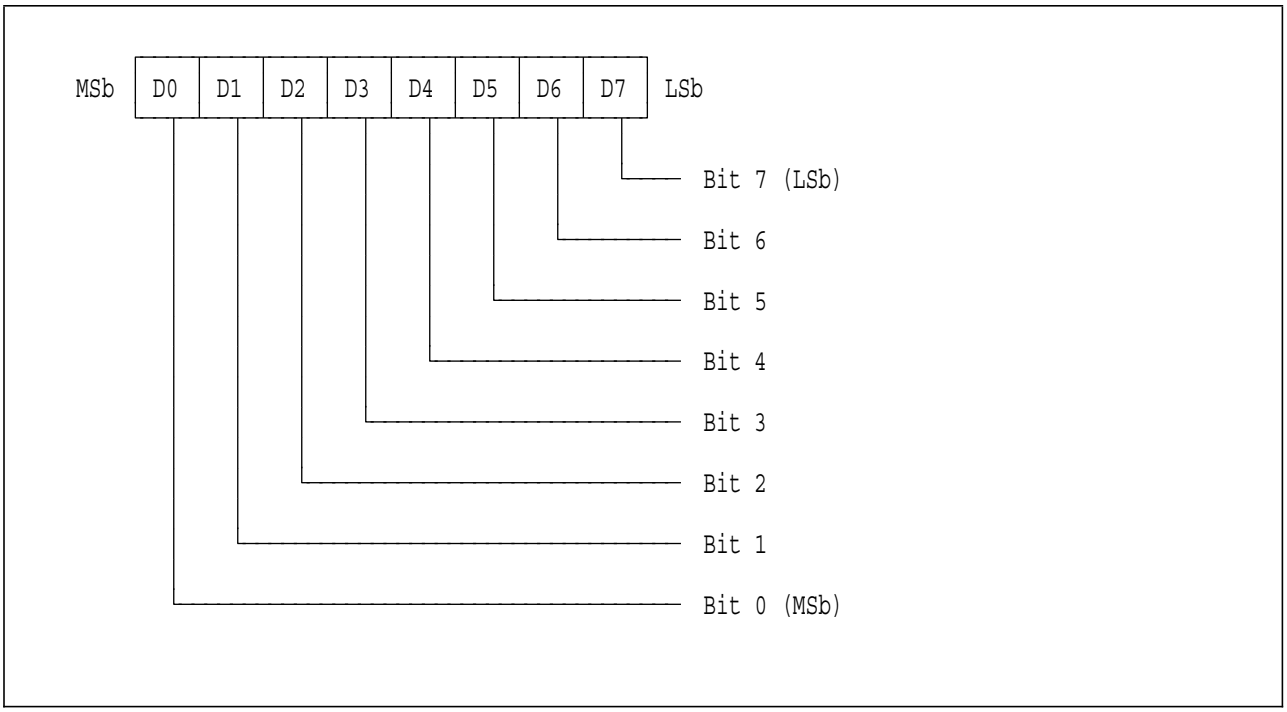


Figure 24. Register Bit Numbering

6.1.5.1 PORT 0092 -- Special Port 92 (Read/Write)

Bit 7: Soft Reset. Changing this bit from 0 to 1 will cause a system soft reset to occur. This bit must be returned to 0 before another soft reset can be issued.

Bit 6: LE Mode. Writing a 1 to this register sets the system board into LE Mode. Writing a 0 changes the system board into BE Mode. A store instruction can be used to change the Endian mode of the system. The PCI Bridge and Memory Controller (e.g. IBM 27 82650) will synchronize the completion of this mode switching before acknowledging the instruction. Software must insure that the switch is complete before information may reliably be transferred into the processor. The example in Figure 26 shows one way of confirming the completion of this switch.

Bits (0:5): Reserved.

6.1.5.2 PORT 0808 -- Hardfile Light Register (Write Only)

Bit 7: Hardfile Activity Light. This bit may be used to control the light that indicates hardfile activity.

0 = Light off.

1 = Light on.

Bits (0:6): Reserved.

This register's initial state after reset is B'xxxx xxx0'. It is reset when port X'4D' in the SIO is utilized to reset the native I/O and ISA slots.

The Reference Implementation system board decodes the following range of addresses for this port: X'0808' - X'080B'.

6.1.5.3 PORT 080C -- Equipment Present Register (Read Only)

Bit 7: L2 Cache Present. This bit indicates whether or not an L2 cache is installed. The bit is 0 when a card in the upgrade slot grounds the correspondingly named signal at the upgrade connector; otherwise, it is 1.

0 = L2 cache installed.

1 = L2 cache absent.

Bit 6: Upgrade Present. This bit indicates whether or not an upgrade processor is installed. The bit is 0 when a card in the upgrade slot grounds the correspondingly named signal at the upgrade connector; otherwise, it is 1.

0 = Upgrade processor installed.

1 = Upgrade processor absent.

/ **Bit 5:** L2 256 KB. This bit indicates whether or not a 256-KB L2 cache is installed. The bit is 0 / when a card in the upgrade slot grounds the correspondingly named signal at the upgrade / connector; otherwise, it is 1.

/ 0 = L2 cache is not 256 KB.

/ 1 = L2 cache is 256 KB or is absent.

/ **Bit 4:** L2 is Copy-Back. This bit indicates whether or not a copy-back L2 cache is installed. The / bit is 0 when a card in the upgrade slot grounds the correspondingly named signal at the / upgrade connector; otherwise, it is 1.

/ 0 = L2 Cache is write-through.

/ 1 = L2 Cache is copy-back or is absent.

Bit 3: PCI Slot 1 Occupied. This bit indicates whether or not there is a card installed in PCI Slot No. 1. The bit is 0 when a card in the slot grounds the correspondingly named signal at the PCI connector; otherwise, it is 1.

0 = Card installed.

1 = Card absent.

Bit 2: PCI Slot 2 Occupied. This bit indicates whether or not there is a card installed in PCI Slot No. 2. The bit is 0 when a card in the slot grounds the correspondingly named signal at the PCI connector; otherwise, it is 1.

0 = Card installed.

1 = Card absent.

Bit 1: SCSI Fuse. This bit indicates the status of the SCSI fuse.

0 = Fuse bad.

1 = Fuse OK.

Bit 0: Reserved.

This register's initial state after reset is "As Populated."

The Reference Implementation system board decodes the following range of addresses for this port: X'080C' - X'080F'.

All of these bits are intended for static reporting. Dynamically changing the corresponding signals at the connectors may have undesirable effects on the operation of the system.

6.1.5.4 PORT 0810 -- Password Protect 1 Register (Write Only)

Bits (0:7): Any Value. Writing any value to this port sets a flip-flop which prevents any subsequent access to addresses X'20' - X'2F' of the Time Of Day Clock NVRAM address space. This flip-flop is cleared only on reset. A read has no effect.

This register's initial state after reset is in unprotected mode. It is reset when port X'4D' in the SIO is utilized to reset the native I/O and ISA slots.

The Reference Implementation system board decodes the following range of addresses for this port: X'0810' - X'0811'.

6.1.5.5 PORT 0812 -- Password Protect 2 Register (Write Only)

Bits (0:7): Any Value. Writing any value to this port sets a flip-flop which prevents any subsequent access to addresses X'30' - X'3F' of the Time Of Day NVRAM Clock address space. This flip-flop is cleared only on power-on reset. A read has no effect.

This register's initial state after reset is unprotected mode. It is reset when port X'4D' in the SIO is utilized to reset the native I/O and ISA slots.

The Reference Implementation system board decodes the following range of addresses for this port: X'0812' - X'0813'.

6.1.5.6 PORT 0814 -- L2 Invalidate Register (Write Only)

Bits (0:7): Any Value. Writing any value to these bits will cause the L2 cache to invalidate all its contents and immediately be in a state to begin caching again. The hardware creates a pulse on the L2_TAG_CLR# line going to the upgrade connector when this port is written. A read has no effect.

Programming Note: To guarantee correct operation of this function, there must be no L2 cache operation immediately following a write to this port. The correct sequence is to disable the cache with a write to Port 081C, invalidate the cache with this port, and enable the cache with a write to Port 081C. *Sync* instructions may be required to assure correct ordering of I/O cycles dealing with L2 cache control.

The Reference Implementation system board decodes the following range of addresses for this port: X'0814' - X'0817'.

6.1.5.7 PORT 0818 -- Reserved for Keylock (Read Only)

This register is reserved for systems which use the keylock function. The register is functional and will indicate the status of any signal connected to the keylock position of the front panel connector.

Bit 7: This bit gives real-time status of the signal wired to the keylock position of the front panel connector.

1 = Signal high.

0 = Signal low.

Bits (0:6): Reserved.

6.1.5.8 PORT 081C -- System Control Register (Read/Write)

Bits (4:7): Reserved.

/ **Bit 3:** Floppy drive motor inhibit. This bit will inhibit the start of the floppy drive motor and is
/ used to eliminate pulsing the motor while doing a device select for media sensing.

/ 0 = Motor signal allowed.

/ 1 = Motor inhibited.

/ **WARNING:** D3 must be set to 0 for correct operation of the floppy drive. It defaults to
/ 0.

Bit 2: Mask Transfer Error. This bit will mask any Transfer Error Acknowledge (TEA) signal going to the processor.

WARNING: Operating with TEA masked defeats all error checking in the system and could lead to lockup on incorrect code or other types of system faults.

0 = TEA masked.

1 = TEA not masked.

Bit 1: L2 Update Inhibit#. Setting this bit to 0 will disable the cache but will NOT invalidate the data tags. Also, no snoop operations are performed and no data updates are made while this bit is a 0. Setting it to a 1 will allow normal caching to resume. See bit 0.

Eieio instructions may have to be utilized to insure correct ordering of operations dealing with L2 cache control.

0 = L2 cache updating is disabled.

1 = L2 cache updating is enabled.

Bit 0: L2 Cache Miss Inhibit#. Setting this bit to 0 allows L2 cache misses to bypass the cache and not update the cache contents. This is useful for data movements that do not require cache updates when old instructions or data should be retained. When this bit is a 1, caching operations are normal. See bit 1.

Eieio instructions may have to be utilized to insure correct ordering of operations dealing with L2 cache control.

0 = Allows bypass.

1 = All L2 misses are updated.

This register's initial state after reset is B'000x xxxx'. It is reset when port X'4D' in the SIO is utilized to reset the native I/O and ISA slots.

The Reference Implementation system board decodes the following range of addresses for this port: X'081C' - X'081F'.

6.1.5.9 PORT 0850 -- I/O Map Type Register (Read/Write)

Bit 7: I/O Map. This bit controls the I/O address mode. Setting this bit to 1 allows a contiguously mapped mode wherein 601 cycles with addresses of 2 GB to 2 GB + 64 KB are contiguously mapped to PCI addresses 0 to 64 KB. Setting this bit to 0 causes discontiguous I/O mode as described in Section 6.1.1, "Processor View of the Memory Map."

/ 0 = Contiguous I/O map mode (601 addresses ISA at 2 GB to 2 GB + 64 KB).

/ 1 = Non-contiguous I/O map mode (601 addresses ISA I/O at 2 GB to 2 GB + 8 MB).

Bits (0:6): Reserved.

The Reference Implementation system board decodes the following range of addresses for this port: X'0850' - X'0853'.

This register's initial state after reset is B'xxxx xxx1' (contiguous mode). The register is reset when port X'4D' in the SIO is utilized to reset the native I/O and ISA slots.

6.1.6 System Interrupt Assignments

Table 15 gives the interrupt assignments for the system interrupts. The table gives the IRQ number, the priority, and the connections. The implementation uses an interrupt controller compatible with two 8259s
/ cascaded through IRQ2. The controller is configured at boot time for interrupt 15 to be level sensitive and
/ contain all the PCI interrupts.

Table 15. System Interrupt Assignments

IRQ	Priority	Connection
0 Master	1	Timer 1 counter 0 (internal to SIO chip)
1	2	Keyboard
2	3-10	Cascade from controller 2
3	11	Com 2, ISA pin B25
4	12	Com 1, ISA pin B24
5	13	ISA pin B23
6	14	Floppy, ISA pin B22
7	15	Parallel LPT 1, ISA pin B21
8 Slave	3	RTC
/ 9	4	ISA pin B04
10	5	Audio, ISA pin D03
/ 11	6	ISA pin D04
12	7	Mouse, ISA pin D05
13	8	SCSI
/ 14	9	ISA pin D07
15	10	PCI interrupts (optional), ISA pin D06

6.1.7 I/O Configuration Space Mapping

Table 16 gives the PCI I/O configuration space mapping. Each I/O device may have up to 256 bytes assigned to configuration registers. The layout of these registers includes more address space than 256 bytes, but enables a simple hardware mapping. An address bit is directly connected to the PCI configuration line, ID SEL. Addresses beyond the limit of 256 bytes will cause more than one device to be selected. Within this table, address lines (i.e. A/D) are numbered in Little-Endian fashion (e.g. 0 is the LSb).

Table 16. I/O Configuration Registers			
Device	**ID SEL Line**	**Processor Address**	**PCI Address**
SIO	A/D 11	8080 0800h-08FF	0080 0800h-08FF
SCSI	A/D 12	8080 1000h-10FF	0080 1000h-10FF
PCI expansion slot 1	A/D 13	8080 2000h-20FF	0080 2000h-20FF
PCI expansion slot 2	A/D 14	8080 4000h-40FF	0080 4000h-40FF
PCI expansion slot 3	A/D 15	8080 8000h-80FF	0080 8000h-80FF
PCI expansion slot 4	A/D 16	8081 0000h-00FF	0081 0000h-00FF
PCI expansion slot 5	A/D 17	8082 0000h-00FF	0082 0000h-00FF
PCI expansion slot 6	A/D 18	8084 0000h-00FF	0084 0000h-00FF
PCI expansion slot 7	A/D 19	8088 0000h-00FF	0088 0000h-00FF

6.1.8 Additional System I/O Mappings

/ Table 17 gives the Reference Implementation processor addresses and the functions of some registers in
/ System I/O space. Four bytes of space are allocated for the interrupt acknowledgement. However, the current implementation need return only 1 byte.

Table 17. Registers in System I/O Space	
Processor Address	**Function**
BFFF EFF0h-EFF3	Memory parity error address
BFFF FFF0h-FFF3	Interrupt vector register

The Memory Parity Addressing will return enough of the parity error address to identify the page in which the parity error exists.

6.1.9 I/O Memory Mapping

Table 18 gives the PCI addresses, corresponding processor addresses and the functions of some registers in I/O Memory.

Table 18. Register Map in I/O Memory		
I/O Memory Address	**Processor Address**	**Function**
3FF0 0100	FFF0 0100	Starting address after hard reset
3FFF FFF0	FFFF FFF0	Flash write address and data
3FFF FFF1	FFFF FFF1	Flash lock out write

6.1.9.1 601-To-ROM Cache Fill Read Cycles (Special Burst)

At power-on time, the 601 comes up in Big-Endian Mode with its cache enabled and begins burst mode fetching at address X'FFF0 0100'. The system board logic, at this time, defaults to Big-Endian Mode and the bring-up code is read.

The ROM is located on the PCI bus physically, but not logically. This requires the PCI Bridge and Memory Controller (PCIB/MC) to decode ROM addresses, run eight cycles to the PCI bus, accumulate the 8 single bytes of read data into an 8-byte buffer line, and control responses to the 601 during the burst cycles.

This cycle is a highly specialized cycle that is not a true burst even though the 601 is in burst mode. It obeys the following rules:

- At the beginning of each cycle, the A/D (2:0) bits are set to zero regardless of the states of the 601_A (29:31) bits.
- Logic in the PCIB/MC causes ROM cycles to run at approximately 150 ns - 200 ns each to accumulate 8 bytes of data. A/D (2:0) are incremented on each cycle. Thus one 8-byte beat takes approximately 1.6 µsec. The timing is 16 CPU clocks on the first beat and 13 CPU clocks on each subsequent beat, with a one- or two-clock overhead after the last beat.
- After the first 8 bytes are accumulated, they are transmitted to the 601 as the first beat of the burst. The same data is repeated on the next three beats in rapid order without cycling the ROM. The reason for this is to prohibit cycles on the PowerPC processor bus which are long enough to interfere with refresh or cause extreme latency.

  Software Develpment Note: The initial stages of bringup must be programmed to account for this pattern of repetition. When enough useful instructions have been executed, the cache may be turned off so that it is not necessary to code the entire Flash in this manner.

- BE/LE data multiplexing follows the same rules as for normal memory, but in this case the low three address bits have no effect on the result. The bytes in Flash 0, 1, 2 ... are passed to 601 byte lanes 0, 1, 2... in BE mode, and they are swapped on byte lanes 7, 6, 5... in LE mode.

6.1.9.2 601-To-ROM Non-Burst Read Cycles

These cycles work very similarly to the special burst cycles described above. They obey basically the same rules. The difference is that they are naturally single-beat cycles, so data is not repeated.

- Any 601-supported-size transfer with any alignment may be used, and the cycle will complete as described in the previous section. Note that 8 bytes of data are always read from ROM.
- BE/LE data multiplexing follows the same rules as for normal memory, but in this case the low three address bits have no effect on the result. The bytes in ROM 0, 1, 2 ... are passed to 601 byte lanes 0, 1, 2 ... in BE mode, and they are swapped on byte lanes 7, 6, 5 ... in LE mode.
- The cycle time is approximately 1.6 µsec to 2 µsec whether 1 or up to 8 bytes are retrieved.

6.1.9.3 601-To-ROM Write Cycles (Flash-Based Implementation)

Writing to Flash ROM is another very specialized cycle. Only one address X'FFFF FFF0' can be used for writing all data to Flash. The Flash ROM address and data are both encoded into the 4 bytes of data written by this store operation. Only a word (4-byte) write cycle is supported (although only 1 byte is written at a time). The first 3 bytes contain the Flash address in order of low-order byte to high-order byte. The fourth byte (most significant byte in the word) contains the byte of data to be placed in the Flash ROM.

Flash ROM protection must be implemented within software. Port X'FFFF FFF1' can be used to lock out all Flash writes. Writing any data to this port address locks out all Flash ROM writes until the power is turned off and back on. In addition, the Flash itself has means to lock out updates to certain sectors by writing control sequences. Consult the Flash chip manufacturer’s specification for details.

6.1.9.4 PCI Masters To ROM

This implementation provides no mechanism for the PCI devices to access the ROM.

6.1.10 DMA Assignments

Table 19 gives the DMA request and grant lines. Details of the DMA controller implementation are provided in Section 6.3, "I/O Complex Components."

Table 19. System Interrupt Assignments

DMA Channel	DRQ	Connection
0	0	I/O connectors
1	1	I/O connectors
2	2	Super I/O, floppy diskette controller
3	3	I/O connectors
4	4	Cascade in SIO
5	5	I/O connectors
6	6	Audio connectors
7	7	Audio connectors

6.2 Processor Complex Components

This section describes components which could be used to implement the processor complex for a PowerPC Reference Platform-compliant system. Refer to Figure 18 for a diagram of the processor complex within the system configuration.

6.2.1 System Board

The system board contains most of the electronics for this Reference Implementation. Major I/O subsystems are connected to main memory through the PCI bus. The graphics subsystem, System I/O and SCSI are located on this bus. The video is attached to the PCI bus through a PCI socket, allowing for future upgradability and a fast interface. Flash ROM accesses share the PCI A/D lines with the control signalling provided by the memory controller.

The board is designed to an industry-standard LBX 9" by 13". It requires + 5 volts to power most of the components. Plus 12 and -12 volts are also required to support some of the peripheral features. Components that require +3.6 volts (601 processor and PCI Bridge/Memory Controller) are supported using a system-board-mounted regulator to convert + 5 to +3.6 volts. Negative 5 volts is supplied to the ISA connectors via the riser.

The system board will be available as a component from the IBM Microelectronics Division.

6.2.2 PowerPC Processor

The 601 processor (IBM PPC601 or Motorola MPC601) is a PowerPC processor available in operating frequencies of 50, 66 or 80 MHz. There are 32 address bits and 64 data bits. It contains a 32-KB eight-way set associated internal cache. The 601 is packaged in a 304-pin quad-flat-pack (QFP) package and requires +3.6-volt power.

6.2.3 PCI Bridge and Memory Controller (PCIB/MC)

This component, IBM 27 82650 (abbreviated as '650) is actually composed of two modules which act together. A buffer chip, IBM 27 82653, makes the connections between the processor, memory and PCI buses under the control of a control chip set, IBM 27 82654. IBM 27 82654 is implemented in a 160-pin chip. IBM 27 82653 is implemented in 304-pin Gate Array QFP chips.

The attributes of the PCI Bridge and Memory Controller are as follows:

- Supports 604 and 603 processors (except the 603 operation in 1:1 clock mode, which is not supported)
- SIMM memory controller
- Bridge from the PowerPC processor to the PCI bus
- Burst or single-beat access from CPU
- Burst or single-beat access from PCI
- PCI and PowerPC processor bus arbitration
- Error reporting to the PowerPC processor
- Supports only non-interleaved memory access operation (paging is supported)
- Memory on the PCI bus is not executable, nor cachable by the processor
- Memory on the ISA bus is not executable, nor cachable by the processor
- Supports Bi-Endian mode switching by performing the appropriate address modification and data multiplexing

The PCIB/MC chip set is available from IBM Microelectronics. Other chip vendors are working on similar solutions.

6.2.4 Memory Subsystem

The Reference Platform's memory subsystem can support up to 256 MB of system memory on eight SIMM sockets. Each SIMM socket can support an 8-MB or 32-MB parity-checking SIMM memory with access speeds less than or equal to 70 ns. These SIMMs are in an industry-standard 168-pin package. These SIMMs are an approved JEDEC standard for 8-byte SIMMs. The DRAM subsystem is 72 bits wide, which includes 64 data bits and 8 parity bits. One parity bit is generated for each byte of data written. During a read operation, one parity bit is checked for each byte of data.

The features and functions of the system memory are as follows:

- Supports parity checking
- Supports 8-MB and 32-MB SIMMs
- Supports mixed use of both types of SIMMs (8 MB and 32 MB)
- Supports empty SIMM sockets at any position

6.2.5 Clock Generation

| The primary clock generation function is accomplished with an IBM clock chip. Sixty-six-MHz systems will
| use the IBM25JP-CLK01 clock chip. Eighty-MHz systems will use the IBM25JP-CLK03 clock chip. The chips are pin compatible and use a seed oscillator to generate the 2X, processor clocks, and 601 bus clocks needed by the system. One of the bus clock outputs feeds a Phase Lock Loop (PLL) clock generator which generates the PCI clocks. In the normal mode of operation the PLL divides the 601 bus clock by 2.

| A second clock chip is used to generate clock signals for the PCI bus. The features and functions of the AMCC S4403 BiCMOS PLL Clock Generator are as follows:

- Generates eight clock outputs for PCI devices
- Forty-four-pin PLCC package

- Maintains +/- 1 ns synchronization to the 601 bus clock and any output clock
- Can be connected for divide-by-1 or divide-by-2 operation

6.2.6 Upgrade Slot

A 2x101 Micro Channel-style card connector socket is provided on the system board in which either an L2 Cache card or a processor upgrade card may be installed. The 601 bus signals and various control signals such as presence detect signals and cache control signals are wired to the socket. Section 6.7, "Upgrade Slot Definition," gives the list of pins and their functions on this connector. Support is included on the system board for:

- Write-through or copy-back cache cards
- Two cache-type ID bits in a system board register
- Various cache controls such as flush and disable
- Powering down the processor installed on the system board when an upgrade processor is detected. (This slot does not support multiprocessing.)
- Arbitration for the upgrade processor

The upgrade processor card must conform to the PowerPC (logical) bus definition. The primary upgrade target is the 604 processor. The upgrade processor card must provide a 3.6- or 3.3-volt regulator as required. Plus 5 volts is available at the socket.

6.2.6.1 L2 Cache

An L2 cache is part of the Reference Implementation. The cache is organized with a 72-bit data/parity path. The cache card will not check parity, but will store and retrieve parity generated elsewhere. The L2 cache for the Reference Implementation will have the following attributes:

- 256 KB or 512 KB
- Pluggable
- Direct-mapped
- 3-1-1-1 burst read hits
- Supports cycle times of 15 ns or greater
- 32-byte line
- Updates cache on bursts and single-beat write hits of exactly 8 bytes
- Invalidates line if single-beat write hit is of less than 8 bytes
- Qualifies cache updates with PowerPC Processor Cache Inhibit signal
- Will provide a minimum of 8 bytes of data on ANY cache read hit
- External L2_CACHE_INHibit Input from System Control Register
- External L2_CACHE_DISable Input from System Control Register
- External L2_CACHE_FLUSH Input from System Control Register
  - This will cause the L2 to invalidate all cache lines; no data will be written back to memory
- Supports storage of parity on data lines
- Caches only lower 2 GB address space (System Memory space)
- No address pipelining is supported

6.2.7 Flash ROM or EPROM

This component of the Reference Implementation contains the bring-up code. In addition, it is recommended that machine model-specific data such as the processor and system board bus speeds, native I/O complement, and memory map boundaries be programmed into this device. There is no other source of this information built into the system board. Programming this data into Flash ROM before or during the manufacturing process is possible.

After power on, the processor fetches the initial code from this device. A maximum of 16 MB ROM space is architected, but the system board supports 512 KB.

A jumper (J20) is provided on the system board which allows use of an EPROM (AMD 27C040-150JC) in place of the Flash ROM (AMD 29F040-120) for systems that do not wish to bear the cost of Flash ROM. Alternatively, a 256-KB EPROM may be utilized.

6.3 I/O Complex Components

This section describes components used to implement the I/O complex for the PowerPC Reference Implementation. Refer to Figure 18 for a diagram of the I/O complex within the system configuration.

6.3.1 Graphics Subsystem

The graphics subsystem consists of a PCI-attached Weitek 9000 and 1 MB of standard video RAM. The VRAM may be expanded to 2 MB. A Brooktree Bt_485 RAMDAC is used to convert digital red-green-blue (RGB) signals to analog video signals.

An alternate video system is the S3 86C928PCI attached to the PCI bus. It contains the same VRAM configuration and RAMDAC as the implementation using the Weitek.

6.3.2 SCSI Subsystem

The SCSI subsystem functions are performed by an NCR 53C810 chip. This component attaches directly to the PCI bus on the system board and is described in detail below:

- Eight-bit SCSI-2 interface
- Supports variable block size and scatter/gather data transfers
- Supports 32-bit word data bursts with variable burst lengths
- Full 32-bit PCI bus master
- Sixty-four-byte FIFO buffer

6.3.3 I/O Control Subsystem

The I/O Control Subsystem is implemented in two chips which are described in the next two subsections.

6.3.3.1 System I/O Bridge

The system I/O bridge function is provided by an Intel 82378ZB. This chip provides a PCI-to-ISA bus bridge where the native I/O and the ISA slots reside. It also provides system services such as DMA timers and interrupt control. Its major functions are as follows:

- Bridge between PCI and ISA
  - Eight- or 16-bit ISA devices
  - Twenty-four-bit addressing on ISA
  - Partially decodes Native I/O addresses
  - Passes unclaimed PCI memory addresses below 16 MB to ISA
  - Passes unclaimed PCI I/O addresses below 64 KB to ISA
  - Generates ISA clock, programmable divide by 3 and 4 ratios
  - Allows ISA mastering and has programmable decodes which map ISA memory cycles to the PCI bus
  - Has 32-bit posted memory write data buffer, no I/O buffering
- Seven-channel DMA between ISA devices and memory

- Eight- or 16-bit devices on ISA (and optionally PCMCIA) bus only
- Thirty-two-bit addressing of DMA
- Function of two 83C37's
- Eight-byte bidirectional buffer for DMA data
- Supports Scatter-Gather DMA operations
- Supports Guaranteed Access Time Mode for DMA and ISA masters

• Timer block -- function of 82C54
• Interrupt controller -- function of two 8259's
• Functions as PCI slave during programming and ISA slave cycles, as bus master during DMA (or ISA master cycles)

6.3.3.2 Native I/O Controller

This component contains the floppy disk controller, two serial ports, the IDE controller, and one parallel port. Control is provided via the ISA bus. This function is provided by a National PC87312 SUPER I/O chip.

Additional hardware and software may be used to facilitate the floppy disk media sense function. The National Super I/O controller chip will normally enable disk rotation anytime the media select line is activated. This pulse to the motor circuitry could lead to component failure and noise when software frequently (e.g. one hundred times a second) polls the floppy media. To eliminate the motor circuit pulse, software that intends to poll the floppy must set the motor inhibit bit in register 81C and hardware must inhibit this signal to the floppy motor circuit. A second approach that does not use hardware is possible for systems that have only one drive. In this approach, software which wishes to sense the media places the controller in four floppy mode and issues a media select for drive zero, but does not issue a motor select for drive zero. To avoid lighting the floppy access lamp, software floppy device drivers should set the drive select bits to B'11' when the floppy is not being used.

6.3.4 Audio Subsystem

Business audio is provided through the Crystal Semiconductor CS4231 stereo audio integrated circuit, which is connected to the ISA bus. The IC provides simultaneous capture and playback with two independently controllable DMA channels, each with its own 16-sample FIFO buffer. In addition, the audio chip provides compression and decompression and Big-Endian and Little-Endian sample modes. The audio subsystem is processor driven and does not include a digital signal processor (DSP). As such, it can play MIDI files, but is not a full-function MIDI system. One audio output is to a single speaker mounted in the cabinet. Also supported are four rear-mounted 3.5 mm jacks for:

• Stereo earphones (the speaker is muted when an earphone plug is inserted into the connector)
• Microphone input
• Stereo line in
• Stereo line out

There is also a system board-mounted connector with cable for direct playback from a CD-ROM.

6.3.4.1 Timer 2 Audio Support

The conventional PC Timer 2 signal is summed with the outputs of the business audio chip at the operational amplifier which drives the speaker. This provides the capability of supporting standard audio on configurations using this system board. Thus the software may elect to directly drive Timer 2 audio or to emulate Timer 2 audio through the business audio chip. Note that the use of the Timer 2 for audio is not recommended.

6.3.5 601 Processor Internal Real-Time Clock

The 601 processor requires a 7.8125-MHz clock input to support its internal Real-Time Clock. All other PowerPC processors contain a Time Base which is driven off of other clocks within the system.

6.3.6 Real-Time Clock (RTC)

Real-Time Clock and calendar functions are performed by a Dallas Semiconductor DS1385S chip. A description of this chip follows:

- RTC Function (PC compatible)
- Four KB Non-volatile RAM
- Connected to the X bus, which is a buffered 8-bit subset of the ISA bus
- Separate replaceable battery
- Sixty-four bytes of CMOS RAM

Figure 25 shows the map of information contained in the CMOS RAM contained on this chip.

6.3.7 Keyboard/Mouse -- Intel 8042AH

An Intel 8042AH chip provides keyboard and mouse controls and is connected to the X bus.

6.3.8 I/O Decoder

This component resides on the X bus. It receives partial decode signals from the SIO chip and further decodes these to produce chip selects for various components. It also contains most of the system registers and implements password protection.

6.4 Endian Switching Process

The process outlined in this section will switch the system from Big-Endian to Little-Endian mode when
/ used in this Reference Implementation with a 601 processor. Because this process must execute during tran-
/ sition periods in which the processor and system components are in different Endian modes, care must be
/ taken to assure that interrupts are not taken or that data and instructions do not remain in cache in the
/ wrong Endian order. The instructions must not cross a page boundary. The process is outlined below:

a) Enable address translation
b) Flush all system caches
c) Disable interrupts
d) Enter supervisor state
e) Set the processor and system board state to Little-Endian (see Figure 26)

Note: Processor is now in Little-Endian mode. All instructions must be in Little-Endian order.

f) Put interrupt handlers and processor data structures in Little-Endian format
g) Enable interrupts
h) Start the Little-Endian operating system initialization

```
/* Structure map for CMOS on PowerPC Reference Platform */
/* CMOS is the 64 bytes of RAM in the DS1385 chip */
/* The CRC's are computed with x**16+x**12+x**5 + 1 polynomial */
/* The clock is kept in 24 hour BCD mode and should be set to UT(GMT) */

#ifndef _CMOS_
#define _CMOS_

struct _CMOS_MAP {
  unsigned char DateAndTime[14]; /* 00 = Seconds
                                    01 = Seconds Alarm
                                    02 = Minutes
                                    03 = Minutes Alarm
                                    04 = Hours
                                    05 = Hours Alarm
                                    06 = Day of Week
                                    07 = Day of Month
                                    08 = Month
                                    09 = Year (two digits)
                                    0A = Status Register A
                                    0B = Status Register B - Alarm
                                    0C = Status Register C - Flags
                                    0D = Status Register D - Battery */
  unsigned char SystemDependentArea1[2];
  unsigned char SystemDependentArea2[8];
  unsigned char FeatureByte0[1];
  unsigned char FeatureByte1[1]; /* 19 = PW Flag;
                                    attribute = write protect */
  unsigned char Century[1]; /* century byte in BCD, e.g. 0x19 currently */
  unsigned char FeatureByte3[1];
  unsigned char FeatureByte4[1];
  unsigned char FeatureByte5[1];
  unsigned char FeatureByte6[1];
  unsigned char FeatureByte7[1]; /* 1F = Alternate PW Flag;
                                    attribute = write protect */
  unsigned char BootPW[14]; /* Power-on password needed to boot system;
                               reset value = 0x00000000000000005a5a5a5a5a5a);
                               attribute = lock */
  unsigned char BootCrc[2]; /* CRC on BootPW */
  unsigned char ConfigPW[14]; /* Configuration Password needed to
                                 change configuration of system;
                                 reset value = 0x00000000000000005a5a5a5a5a5a);
                                 attribute = lock */
  unsigned char ConfigCrc[2]; /* CRC on ConfigPW */
  } CMOS_MAP;

#endif  //ndef _CMOS_
```

Figure 25. Map of CMOS on DS1385S

```
    x00  mfspr     R2,1008    ;Load the HID0 register
    x04  ori  R2,R2,LE_BIT    ;Set the Little-Endian bit in R2
    x08  sync
    x0C  sync
    x10  sync
    x14  mtspr     1008,R2    ;Set the processor into Little-Endian mode
  ;At least 3 sync instructions must precede and follow the above
  ;instruction because of processor design and pipelines.
    x18  sync
    x1C  sync
    x20  sync
    x24  stb  R5,0(R29)  ;Set Endian mode of system board
/ ;Register R5 has the data and R29 has the address for the Endian control port
  ;Endian control port at X'8000 0092' must be addressed at
  ;X'8000 0095' because processor is modifying addresses now
  ; To this point all instructions are in Big Endian Format
/ ;Include a string of palindromic instructions to pass time until the system
/ ;completes the switch.  Twenty five are suggested based on a 66 MHz processor
/    x2C  addi  R0,R1,0x138     ;Instructions which work LE or BE
/                    ;This instruction generates 0x38010138
/    x30  addi  R0,R1,0x138
/    x34  addi  R0,R1,0x138
/    ...
/    xyy  addi  R0,R1,0x138
  ;Start of Little Endian instructions
```

Figure 26. Instruction Stream to Switch Endian Modes

/ This switching process is only applicable for 601 processors in a design with a Bi-Endian memory as shown / in Figure 56.

/ If the design uses the Big-Endian memory approach as shown in Figure 57, then this process must be modi- / fied. The Little-Endian portion of the operating system loader and any of the rest of the operating system / which was brought into memory before the system was switched to Little-Endian mode does not reside in / memory in the correct format. This information was brought into memory in the true Little-Endian format / as it existed on the media. It must exist in memory in the “PowerPC Little-Endian” form. This form has / the bytes reversed within each doubleword. After the system is in Little-Endian mode, further input from / media of Little-Endian information will arrive in system memory in the correct form. Examples of / approaches to accomplish this byte reversal are as follows:

/ • Before the processor switches to Little-Endian mode, load and store each Little-Endian doubleword / doing a byte reversal in the process.
/ • Swap out the Little-Endian material while in the Big-Endian mode and swap it back in after the system / is in Little-Endian mode and before the processor has been switched to Little-Endian mode. The / instructions to perform the swap-in must have been byte reversed.
/ • During the install process, byte reverse each doubleword of Little-Endian information on the media. / This format cannot be moved to an implementation with Bi-Endian memory.

6.5 Devices and Subsystems Used

The list of devices described below is intended to reflect the typical devices used in the PowerPC Reference Platform Implementation. No endorsement of the product or vendor is stated by this list. This list is a minimum set of devices requiring operating system support. These devices, adaptors, and subsystems help the Reference Implementation compete with other PCs and workstations. These system solutions establish the recommended implementation. Alternative devices may be used in compliant implementations, but any software-visible difference will have to be accounted for by the vendor in the abstraction software.

6.5.1 Storage Subsystems

Hard Disk Drives (SCSI-2)

Vendor	Description
Quantum	270-MB Hard Disk Drive
Conner	340-MB Hard Disk Drive
Conner or Maxtor	540-MB Hard Disk Drive
IBM SSD	1-GB Hard Disk Drive
IBM SSD	2-GB Hard Disk Drive

Floppy Disk Drives

Vendor	Description
Alps	1.44-MB 3.5" FDD
Alps	2.88-MB 3.5" FDD
Canon	1.22-MB 5.25" FDD

CD-ROM Drives (SCSI-2)

Vendor	Description
Toshiba	CD-ROM XA DS

Optical Disk Drives

Vendor	Description
IBM SSD	M-O Optical drive 3.5" 128 MB 40 ms

Tape Drives (SCSI)

Vendor	Description
HP	35470A DAT
HP	35480 DAT
WangDAT	3200
Wangtek	5525ES (QIC)
Wangtek	51000ES (QIC)
Tandberg	3820
Tandberg	4120

Disk Arrays (SCSI)

Vendor	Description
Microarray	RAID5 Disk Array
OASIS	RAID5 Disk Array

6.5.2 ISA Bus Subsystems

Communications

Vendor	Description
IBM	Token Ring Network 16/4 Adaptor
IBM	PS/VP 10BASE-T Ethernet Adaptor
IBM	3278/79 Emulation Adaptor
IBM	Enhanced 5250 Emulation Adaptor Kit
IBM	X.25 Adaptor
3COM	3Com EtherLink** ISA Adaptor
Intel	EtherExpress** 16 (ISA)
Novell	NE2000/3200 (ISA)
Ungermann-Bass	NIUPS
Standard MicroSystems	EtherCard** (ISA)

MultiMedia Adaptors

Vendor	Description
IBM	Mwave Adaptor
Creative Labs	Sound Blaster** Pro/16

Fax/Data Modems

Vendor	Description
Hayes	Modem 2400-14400
Intel	SatisFAXtion** Modem
Megahertz	Modems
Practical Peripherals	Modems
US Robotics	Modems

Printer/Plotter Devices

The following types of printers/plotters should be supported:

- EPSON Dot Matrix
- HP (PCL)
- HP (Plotter)
- PostScript** compatible

Scanners

Vendor	Description
HP	ScanJet** IIp/c
(Any)	Scanners with TWAIN-standard interfaces

6.5.3 Human Interface Subsystems

Display Options

The following types of displays are supported:

- CRTs capable of 1024x768 and 1280x1024
- LCDs capable of 640x480 and 1024x768

PS/2-Compatible Mouse

Vendor	Description
IBM	3 button
IBM	2 button

6.5.4 PCI Subsystems

Vendor	Description
S3	Graphics Adaptor
Weitek	Graphics Adaptor
IBM GTX	Graphics Adaptor

6.6 Base Configuration and Capacities

This section contains lists of components for three example desktop configurations as well as a list of components common to all three. These lists are given to provide a range of system configurations or models possible with the Reference Implementation.

- Common to all models
  - Power Supply
  - 1.44-MB Diskette Drive
  - 5.25" CD-ROM
  - System Board
  - Keyboard
  - Mouse
- Model 1
  - 8-MB SIMM
  - Hard drive -- 240 MB
  - Base Video (1-MB VRAM, PCI attached, 1024x768x8 bit)
- Model 2
  - 2 x 8-MB SIMM (16 MB)
  - Hard Drive -- 340 MB
  - Cache Card -- 256 KB
  - Base Video (1 MB, 1024x768x8 bit)
- Model 3
  - 32-MB SIMM
  - Hard Drive -- 540 MB
  - Cache Card -- 256 KB
  - Extended Video (2 MB, 1280x1024x8 bit)

6.7 Upgrade Slot Definition

The upgrade slot consists of a 2x101 Micro Channel-style card edge connector. The slot is designed to support a write-through L2 cache, a copy-back L2 cache, or an upgrade processor (currently targeted for a 604). If both an upgrade card and an L2 card are required, it is possible to design a short riser to accept both; however, the electrical performance of the local bus degrades due to loading and reflections if this is done. The Reference Implementation currently does not support both.

This section defines the detail for the upgrade slot of the Reference Implementation. The definition of the signals provided on the upgrade slot is enumerated. The protocol for both write-through (WT) and copy-back (CB) secondary caches is defined.

6.7.1 Upgrade Slot Signal Descriptions

The following table describes the functions of the signals included at the upgrade slot connector:

Table 20 (Page 1 of 5). Upgrade Connector Signal Definitions

Signal Name	Pin Type	Description
PROCESSOR BUS SIGNALS: The pin type is stated from the point of view of the option card for CB or WT cache cards. Upgrade cards drive all signals listed in this group.	--	See PowerPC processor-specific user's manuals
60X_A(0:31)	Bi	CPU address bus, 0=most significant bit The '650 bridge drives the 60X_Address lines during an I/O snoop operation. (CPU address parity is not supported)
60X_D(0:63)	Bi	CPU data bus, 0=most significant bit Note that the '650 bridge does not drive data on the 60X_Data lines during an I/O snoop operation.
60X_D_PARITY(0:7)	Bi	CPU data parity, 0=most significant bit Parity is present on these lines when the PowerPC processor provides data and when the memory provides data to the PowerPC processor. Cache devices must store and forward the parity or run the system with all error checking disabled. (Mask_TEA bit set in System Control Register, '081C'.) Note that the '650 bridge does not drive data on the 60X_Parity lines during an I/O snoop operation.
TT(0:4)	Bi	CPU transfer type During an I/O snoop the '650 bridge drives these lines to '01010'b for read and '00010'b for write.
TSIZ(0:2)	Bi	CPU transfer size During an I/O snoop the '650 bridge drives these lines to '100'b.
TC(0:1)	In	CPU transfer code During an I/O snoop these lines float.
CI#	In	CPU cache inhibit During an I/O snoop this line floats.
GBL#	In	CPU global This signal is pulled to ground through a 1-K resistor on the system board.

Table 20 (Page 2 of 5). Upgrade Connector Signal Definitions		
Signal Name	**Pin Type**	**Description**
WT#	In	CPU write-through During an I/O snoop this line floats.
SHD#	Bi	CPU shared This signal is pulled high with a 10-K resistor on the system board.
TBST#	Bi	CPU transfer burst During an I/O snoop the ’650 Bridge drives this line to '1'b.
TS#	Bi	CPU transfer start The ’650 bridge drives this line active for one clock period to initiate an I/O snoop.
XATS#	Bi	CPU extended transfer start (PIO) (Not supported on system board.)
AACK#	Bi	CPU address acknowledge Cache cards who claim a cycle must pace the transfer according to the protocol in this document. If the card drives this signal active (low), it must drive it high for one clock before tri-stating the signal. The ’650 bridge drives this signal low for one clock period to terminate an I/O snoop cycle.
ARTRY#	Bi	CPU address retry Cache cards drive this signal low on an I/O snoop hit. See Section 6.7.2, “Protocol for Copy-Back and Write-Through Secondary Caches,” for details.
DRTRY#	Bi	CPU data retry This line is driven low whenever the ’650 bridge drives the TEA signal. During an I/O snoop the ’650 bridge does not drive this line.
TA#	Bi	CPU transfer acknowledge Cache devices who claim a cycle must pace the transfer according to the protocol in this document. If the card drives this signal active (low), it must drive it high for one clock before tri-stating the signal.
TEA#	In	CPU transfer error acknowledge If the ’650 detects a DPE parity error signal from the processor following a sequence in which an L2 cache provides data to the processor, it will activate the error protocol and drive “TEA.”
BR_60X#	In	CPU bus request

Table 20 (Page 3 of 5). Upgrade Connector Signal Definitions		
Signal Name	**Pin Type**	**Description**
BG_60X#	In	CPU bus grant Cache cards may monitor this signal to determine if a cycle originates with the CPU or with the ’650 Bridge.
DPE#	In	CPU data parity error
SYSTEM BOARD INTER-FACE SIGNALS: The pin types in this section are stated from the point of view of the option card.		
HRESET#	In	Hard Reset Output from system board; active low during power-on reset or when reset button is pressed. This signal should be wired to the HRESET# of upgrade processors.
BUS_CLK(0:2)	In	Bus clock Three copies of the bus clock signal. These run at 66 MHz in the Reference Implementation. The rising edge of each is within 0-800 ps after the rising edge of 2X_Clk.
FULL_SPEED	Out	Full speed Strapping this signal to ground will cause the system board to operate at half speed on the local bus. This may be used for debug purposes or to force the local bus to operate at one-half of the processor frequency. It may be necessary to change the configuration of the PCI clock generator if this signal is grounded. It has a pullup on the system board.
BR_L2#	Out	Bus request L2 This signal is wired to the ’650 Bridge to support copy-back L2 cache cards. Its priority is after the PowerPC processor. See Section 6.7.2.6, “Copy-Back L2 (CB L2) Protocol,” for details.
BG_L2#	In	Bus grant L2 This signal is wired to the ’650 Bridge to support copy-back L2 cache cards. See Section 6.7.2.6, “Copy-Back L2 (CB L2) Protocol,” for details.
L2_CACHE_PRESENT#	Out	L2 cache present This signal should be strapped to ground on any L2 cache card. Its state is reported in the equipment register '080C'. It also modifies the behavior of the ’650 Bridge chip set when it is low. It is not intended to be dynamically changed.

Table 20 (Page 4 of 5). Upgrade Connector Signal Definitions

Signal Name	Pin Type	Description
L2_CACHE_CB	Out	L2 cache copy-back This signal should be left unconnected for copy-back and strapped to ground for L2 cache cards that are write-through. Its state is reported in the equipment register '080C' for diagnostic or other software purposes. Its state does not modify the behavior of the ’650 Bridge chip set. It is not intended to be dynamically changed.
L2_CACHE_256K	Out	L2 cache 256 KB This signal is strapped to ground for L2 cache cards that are not 256 KB. Its state is reported in the equipment register '080C' for diagnostic or other software purposes. It does not modify the behavior of the ’650 Bridge chip set when it is low. It is not intended to be dynamically changed.
L2_CLAIM#	Out	L2 cache claim This signal is driven by an L2 cache card to indicate that it will provide or accept data. The ’650 Bridge defers the memory cycle to the L2 when this signal is activated according to the protocol set out in this document. It must be synchronous to the BUS_CLKs at the connector. The state of L2_CLAIM# while TS# is active and in the next clock cycle is immaterial. It is sampled in the second clock interval after the clock interval in which TS# is active and thereafter during the same bus transaction. Its state (high or low) must be maintained until AACK# is asserted by the option card or by the system processor.
L2_TAG_CLR#	In	L2 tag clear This signal is output from the register at port '0814'h. As long as this signal is low, the L2 cache should reset all tags to invalid and stay in a reset condition.
L2_UPDATE_INH#	In	L2 update inhibit This signal is output from bit 1 of the register at port '081C'h. As long as this signal is low, the L2 cache should disable all cache functions including snooping, but it should not invalidate any tags.
L2_MISS_INH#	In	L2 miss inhibit This signal is output from bit 0 of the register at port '081C'h. As long as this signal is low, the L2 cache should inhibit updating the cache contents on a miss.

Table 20 (Page 5 of 5). Upgrade Connector Signal Definitions		
Signal Name	**Pin Type**	**Description**
UPGRADE_PROC_PRES#	Out	Upgrade processor present This signal should be strapped to ground on cards having an upgrade processor. A low on this signal causes the 601 processor to tri-state all outputs and enter a low-power mode. The system expects this to be a static signal and it may not be changed dynamically. The state of the bit is reported in the Equipment register at port '080C'.
BUS_CLK_SPEED(1:0)	Out	Bus clock speed These signals are used by the system board to pass information about the CPU bus speed to the upgrade card so that the upgrade card may select an appropriate clock multiplier. For example, if the bus speed indicated is 66 MHz and an upgrade processor is capable of 99-MHz operation, the card may select a 1.5x multiplier. If the processor were capable of 132 MHz, it would select a 2.0x multiplier. The coding is as follows: • 00 = 66 MHz • 01 = Not presently defined • 10 = Not presently defined • 11 = Not presently defined The Reference Implementation connects these pins to ground (00b). Cards which do not require bus clock speed information may tie these pins to ground.
INT_60X#	In	CPU Interrupt This signal originates at the ’650 bridge. It is included for upgrade processor support.
SRESET#	In	CPU soft reset This signal should be wired to the SRESET# pin of upgrade processors.
GROUND	--	There are 28 ground pins.
+5 VOLTS	--	There are 20 +5-volt pins. Note that devices requiring other supply voltages will require regulators. The current capacity at 0.5 amp per pin is approximately 10 amps. (The system power budget may be less.)
RESERVED	--	There are seven reserved signals at the connector. These are no-connects on the system board.

Figure 27 defines the timing relationships for the upgrade slot. The characters enclosed in brackets, < > , refer to the table values found in Table 21. The figure does not represent any particular operation.

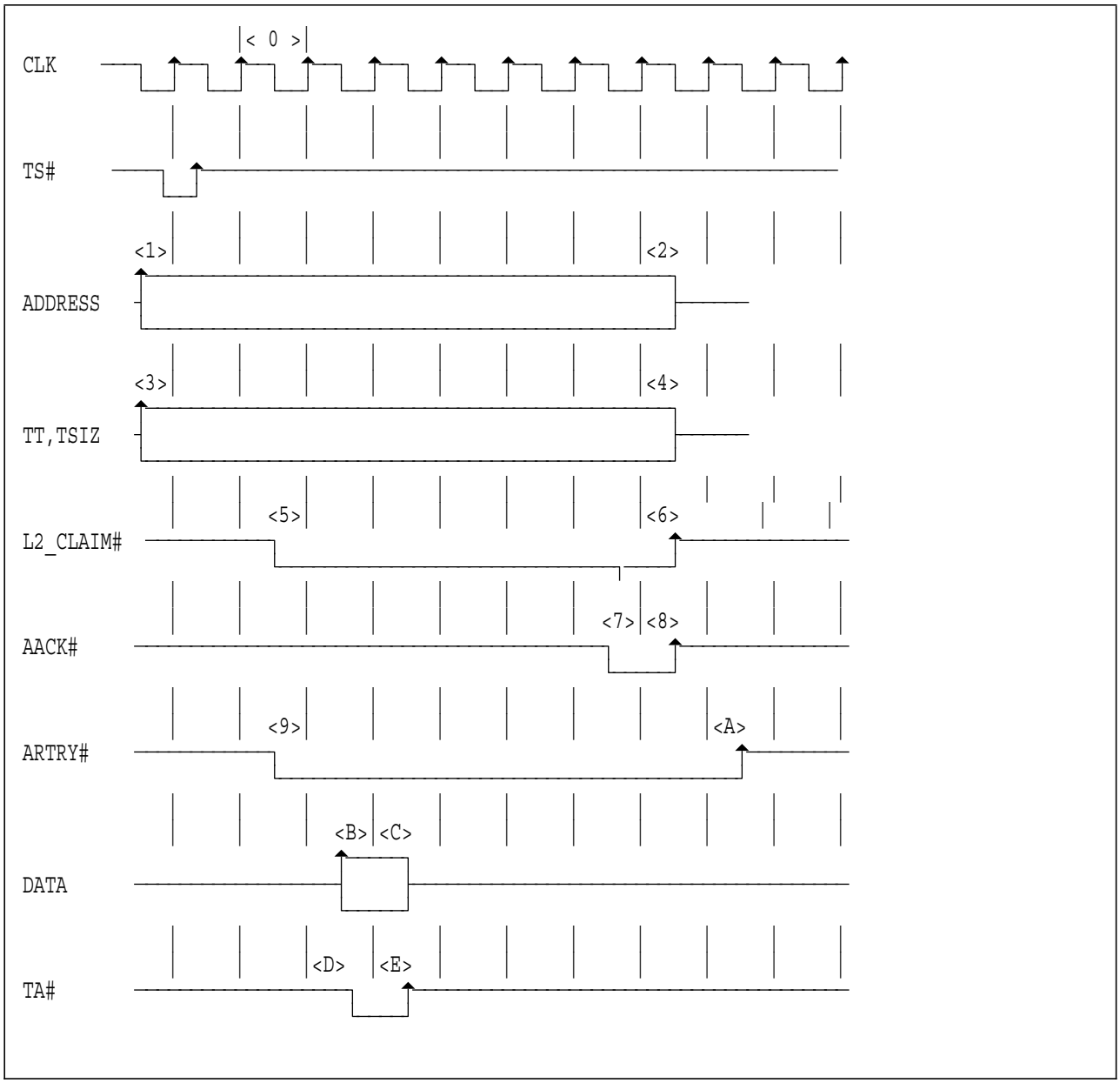


Figure 27. Upgrade Slot Synchronous Signal Timings

Table 21. Upgrade Slot Synchronous Signal Timing and Load Parameters

< >	SIGNAL	PARAMETER	RECEIVE (ns)	DRIVE (ns)	MAX LOAD (pf)	MAX LENGTH (inch)
0	BUS_CLK	CYCLE TIMING	n/a	15	13	1
0		DUTY CYCLE %	n/a	40/50%	--	--
1	60X_A(0:31)	SETUP	2	6	50/10	2
2		HOLD	0	0	--	--
3	TT,TSIZ	SETUP	2	6	50/10	2
4		HOLD	0	0	--	--
5	L2_CLAIM#	SETUP	n/a	6	n/a	n/a
6		HOLD	n/a	0	--	--
7	AACK#	SETUP	2	6	50/10	2
8		HOLD	0	0	n/a	--
9	ARTRY#	SETUP	2	6	50/10	2
A		HOLD	0	0	--	--
B	60X_D(0:63)	SETUP	2.5	6	50/10	2
B	60X_D_PARITY(0:7)	SETUP	2.5	6	50/10	2
C		HOLD	0	0	--	--
D	TA#	SETUP	2	6	50/10	2
E		HOLD	0	0	--	--

Note:

- In the Maximum Load column, the first number is load that is driven by the option card; the second number is maximum load allowed on the option card including wire and pins.
- All signals are synchronous with respect to the BUS_CLK.
- The figures in the Drive column indicate the setup and hold times that the option card must meet at the connector interface.
- The figures in the Receive column indicate the set up and hold times that the system board provides with the specified load.

Table 22 shows the timing and load requirements for the asynchronous L2 cache control signals at the upgrade slot. These signals are asynchronous with respect to the BUS_CLK. However, they are stable at least 40 ns prior to any TS#. In the Reference Implementation, they transition approximately 50 ns prior to the end of a memory-mapped 601-to-PCI I/O cycle. L2_TAG_CLR# is a pulse of approximately 500 ns length which returns to the high state approximately 40 ns before any possible TS#.

Table 22. L2 Control Signal Timing and Load Parameters

SIGNAL	SETUP (RE:TS#)	MIN PULSE	NOM PULSE	MAX LOAD	MAX LENGTH
L2_UPDATE_INH#	40 ns	S/W	S/W	50 pf	8"
L2_TAG_CLR#	40 ns	120 ns	500 ns	50 pf	8"
L2_MISS_INH#	40 ns	S/W	S/W	50 pf	8"

Note:

- MIN refers to the minimum pulse width allowable by the L2 design.
- S/W indicates that the actual pulse width is controlled by software because the signals are outputs of registers contained on the system board.
- The loads and line lengths are the maximum allowed for a device in the slot.

The Reference Implementation L2 cache design specifies that software must activate L2_UPDATE_INH# by writing to its control register, then activate L2_TAG_CLR# (which produces a pulse), and then deactivate L2_UPDATE_INH#. This gives approximately 500 ns minimum timing between edges as shown in Figure 28; however, to accommodate future enhancements, it is recommended that designers assume 120 ns minimum.

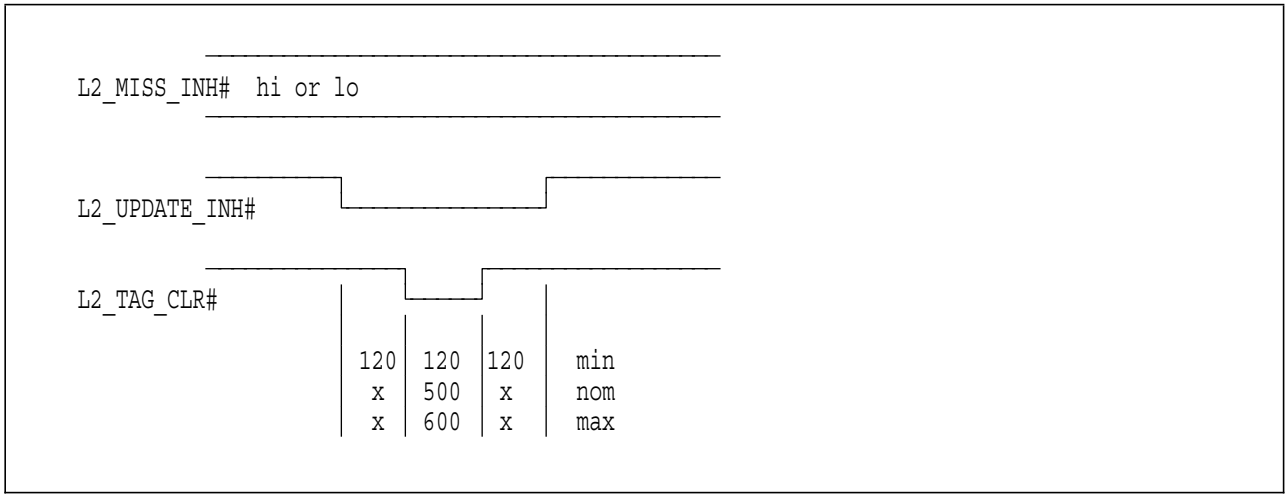


Figure 28. L2 Control Signal Timings

The following outputs from the upgrade card may be strapped to ground or not connected.

- L2_CACHE_PRES#
- L2_CACHE_CB
- L2_CACHE_256K
- L2_PROC_PRES#

6.7.2 Protocol for Copy-Back and Write-Through Secondary Caches

This section will describe the required protocol for any L2 designed to work in the upgrade slot of this system. The bridge chips used are the IBM27-82650 bridge chip set.

6.7.2.1 Bridge Chip Functions

The IBM 27 82650 Bridge chip set ('650) is responsible for system memory cycles as well as generation of I/O snoop cycles to maintain cache coherency in the L1 (PowerPC processor) and L2. The signal at the slot called L2_CACHE_PRESENT# must be continuously asserted (grounded) if the optional L2 is installed.

For PowerPC processor-to-memory cycles, the '650 will sample the L2_CLAIM# input the second clock after TS# is asserted by the processor. If L2_CLAIM# is not asserted and L2_CACHE_PRESENT# is asserted, the '650 Bridge will pace both the data and address tenures to the PowerPC processor. This means that the '650 Bridge will drive all handshaking signals (AACK#, ARTRY#, TA#, etc.) as well as provide the data from memory. The '650 Bridge will assert AACK# on the last data transfer cycle (be it a single-beat cycle or 4-beat burst), coincident with the last TA#. See Figure 29.

Note: The '650 Bridge will sample L2_CLAIM# only if L2_CACHE_PRESENT# is asserted.

On PCI-to-memory cycles, the '650 Bridge will master the PowerPC processor bus to drive the required snoop cycles to the processor and L2. Contrary to I/O PowerPC processor-mastered cycles, the '650 Bridge will not sample the L2_CLAIM# line. The '650 Bridge will drive AACK# active the clock after TS# is asserted, and sample ARTRY# the third clock. Also, it should be noted that the '650 does not drive data on to the 60X_Data lines during an I/O snoop.

The '650 Bridge will sample ARTRY# asserted the second clock after TS# to determine if there was a snoop hit in either the L1 or the L2. If there is a hit, the PCI device is signalled to retry and the PCI bus grant is removed.

Note: The '650 Bridge will NOT drive TA on to the PowerPC processor bus on I/O snoop cycles. It drives TBST# high. The transfer type encoding output during an I/O snoop will be:

PCI READ from Memory	TT(0:4) = 01010
PCI WRITE to Memory	TT(0:4) = 00010
	The '650 Bridge does not drive or sample TT4. This is pulled down on the system board.

If the '650 Bridge samples ARTRY# active on the second clock after TS#, it will drive ARTRY# inactive the second clock after AACK# is inactive, and then tri-state its buffer. This is required, since neither the L2 nor the PowerPC processor can restore this signal -- they both may be driving it, and one is a 3.x-volt part while the other may be a 5-volt part. Note that the '650 Bridge must be configured during the set-up process to enable this function.

The '650 Bridge will sample L2_BR# and 60X_BR# the clock after ARTRY# was asserted (third clock after TS#) by either one or both the PowerPC processor and L2. If only one is active, the '650 Bridge will grant the bus to that requester before granting the bus to another master (with the possible exception of refresh).

If, on the clock after ARTRY# was asserted to the '650 Bridge, both the PowerPC processor and L2 bus request lines are active, the '650 Bridge will grant the bus to the processor. If after the end of the cycle the L2_BR# is still active, then the '650 Bridge will grant the bus to the L2. However, the copy-back L2 will see a "write with kill" cycle when the processor pushes its data to the memory, and the CB L2 cache will invalidate or update its line and drop its bus request. The data from the processor is always more current than the L2 data.

The '650 Bridge will sample DPE# from the PowerPC processor two clocks after each TA# asserted by the L2 (i.e. L2_CLAIM# was asserted) to determine if the L2 is functioning correctly. If the '650 Bridge samples DPE# asserted, then TEA# will be asserted if the cycle is still in progress on the bus. Because of this function, all L2 caches must store and forward parity or the system must be operated with all error checking disabled.

6.7.2.2 General L2 Controller Requirements

L2 caches must manage coherency on a 32-byte line basis.

This protocol requires that L2_CLAIM# be valid the second clock after TS. This means that any cache designed for use in this system must be able to decode a cache hit within this time and assert this line. L2_CLAIM# may be asserted before the second clock cycle after TS, but must be valid on this clock cycle. It must be held active from the second clock cycle after TS# through the clock cycle in which AACK# is asserted.

If L2_CLAIM# is driven active as mentioned above, the L2 cache has claimed the cycle. This means it is responsible for pacing both the address phase (with AACK#) and the data phase (with TA#).

An L2 controller that must drive ARTRY# must be able to drive this signal on the third clock after TS# and must leave it asserted until the clock after AACK# is de-asserted.

The following are required of any L2 used in the system:

- Assert L2_CACHE_PRESENT# (tie low, static).
- If DISABLE_L2_CACHE# is asserted, the cache is essentially turned off.

  No snoops or updates are performed. Only two operations will be performed if the cache is in the disable state:

  a) Power-On Reset
  b) Reset Tags
- If L2_MISS_INH# is asserted, the cache performs I/O snoops only.

  No replacements are done in the cache on a miss. Cache responds to hits only by invalidation, or by providing the data.
- If L2_TAG_CLR# is asserted, the cache will reset itself and clear all data valid bits. No copy-back of dirty data is required.
- Determine if the cycle is a PowerPC processor-mastered cycle, or an I/O snoop cycle generated by the ’650 Bridge.
- Assert L2_CLAIM# the second clock after TS# is asserted on PowerPC processor-mastered cycles.
- Assert ARTRY# (if required) the second clock after TS# is asserted.
- Maintain coherency on 32-byte cache lines (sectors) granularity.
- Store data parity.

6.7.2.3 Determining I/O (PCI) Snoop Cycles

The CB L2 must be able to distinguish between a PowerPC processor cycle to or from memory and an I/O snoop cycle because no data is present on the processor data lines on an I/O snoop cycle. The CB L2 can determine the master of the current data tenure by sampling the BG_60X# on the clock before TS# is sampled active. This is required, since the arbiter can remove the PowerPC processor bus grant on the clock in which TS# is active.

6.7.2.4 Write-Through L2 (WT L2) Protocol

A WT L2 is used to provide the PowerPC processor faster read access to memory when any reads miss the L1. The L2 does not provide any performance improvement on cache cast-outs (PowerPC processor- to-memory writes).

Coherency is enforced by the I protocol. That is, data in the L2 can only be in one of two states:

a) Valid
b) Invalid

The data can only be changed from invalid to valid on burst reads or writes from the PowerPC processor. The WT L2 will sample the CI# output of the PowerPC processor. If CI# is asserted (CACHE_INHIBIT), no caching will occur on misses. If the data is already present in the WT L2, the state of the CI# input is ignored.

Data is invalidated for any of the following reasons:

- Any write of less than 8 bytes to a cached sector
- L2_TAG_CLR# is asserted

Since data is never stale or dirty in the WT L2, any hit in the cache to a valid sector will simply overwrite the previously stored data on a write cycle, and produce the data on a read cycle. This means that if the data is written by the PowerPC processor as a write with kill, the data can still be stored in the WT L2, but will be invalidated on an I/O snoop of a PCI write.

The WT L2 snoops all transactions on the PowerPC processor bus to maintain coherency, as well as to determine whether or not it can provide the data or should store the data.

The WT L2 ignores all address-only cycles (TT3=0). Note that ignoring address-only cycles does not violate the PowerPC architecture because in this implementation there is no cachable I/O memory.

On WT L2 cache hits (read only), the L2 will always assert L2_CLAIM# on the second clock cycle after TS# is sampled low. On reads, it will assert TA# the same clock as L2_CLAIM#, and on burst accesses, will assert a TA# on the next three clocks. See Figure 30, Figure 31, and Figure 32 for details.

6.7.2.5 Write-Through L2 Timing Diagrams

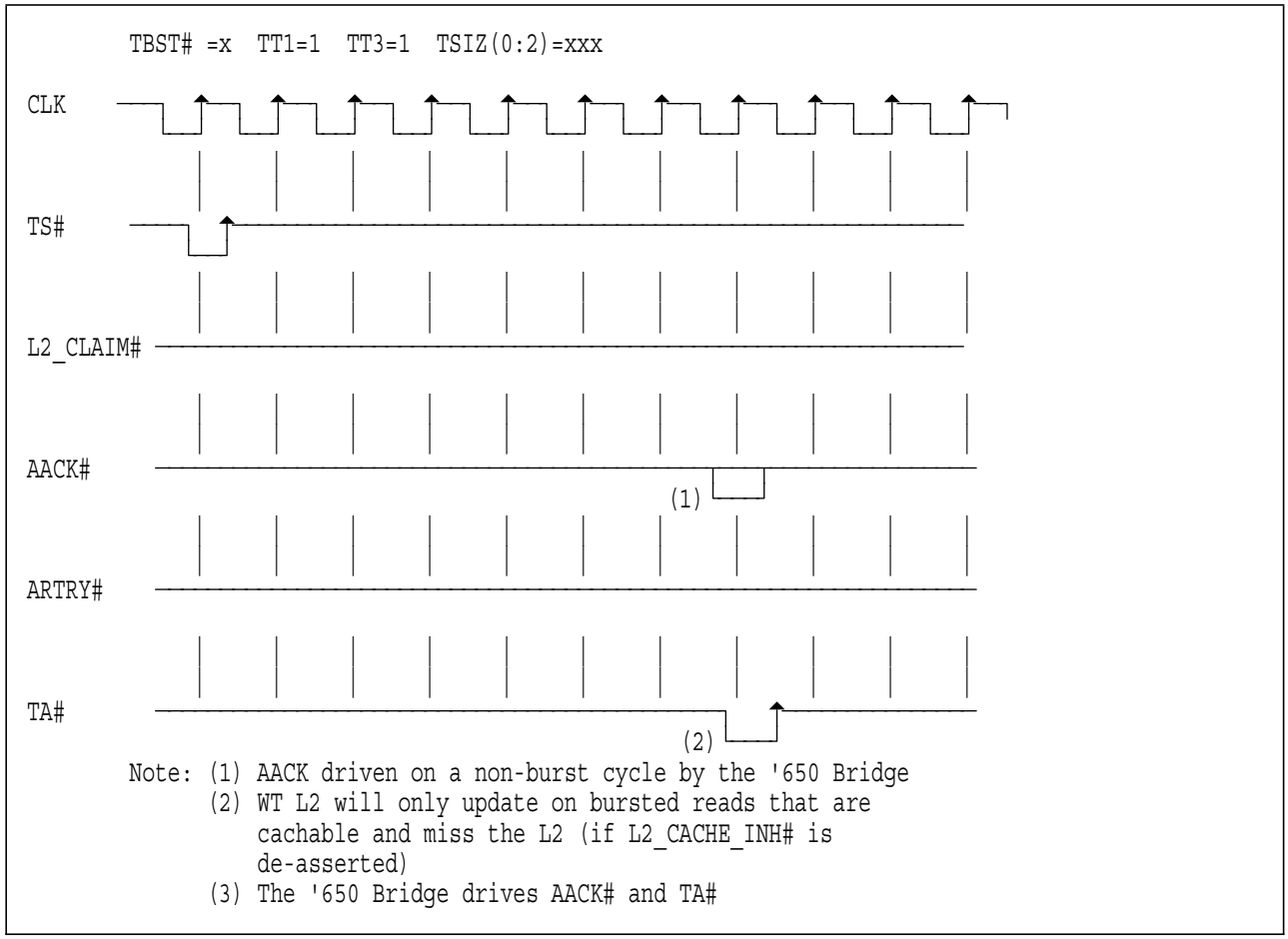


Figure 29. WT L2 Response to Read Misses

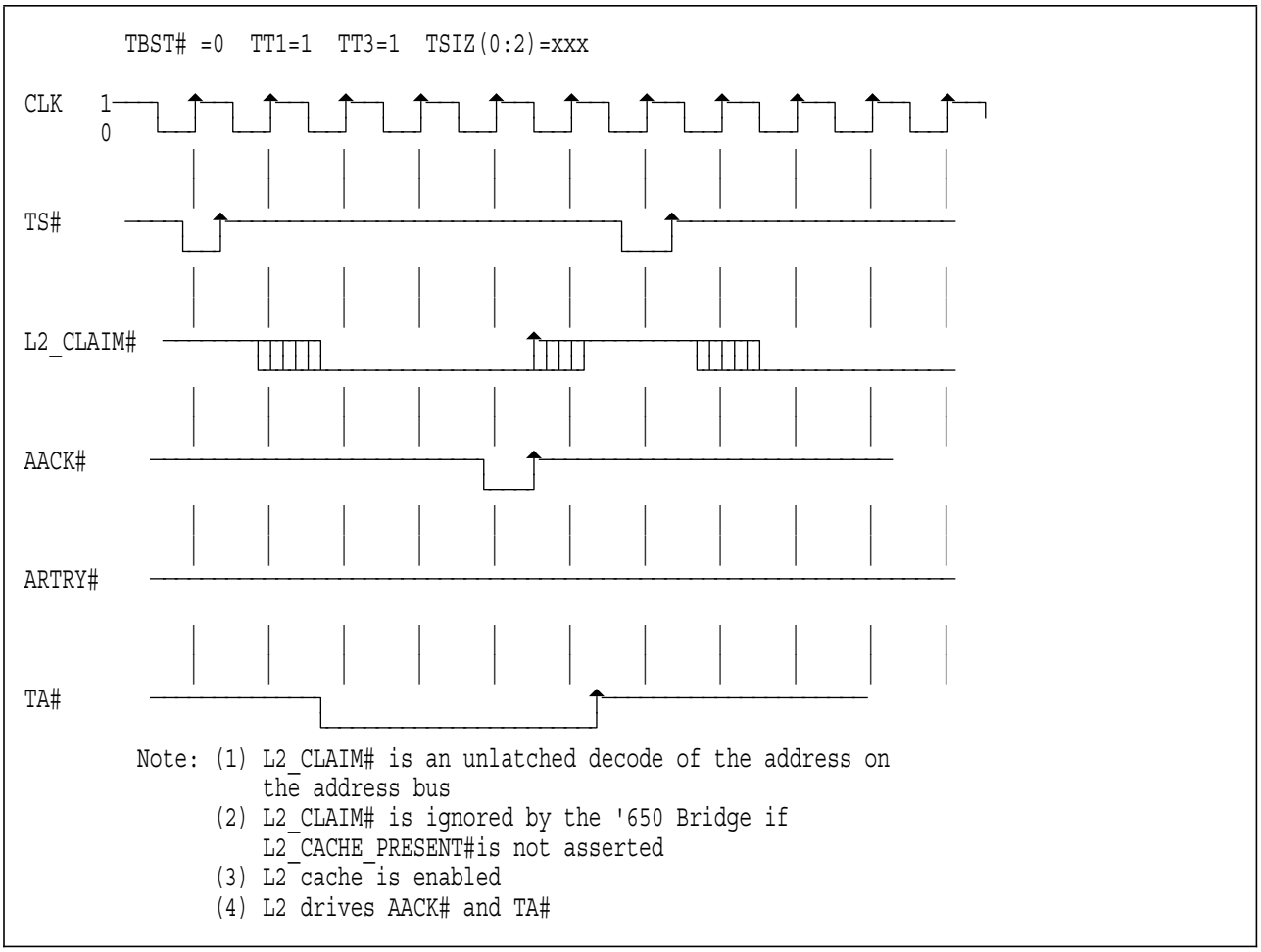


Figure 30. WT L2 Response to Burst READ Hits

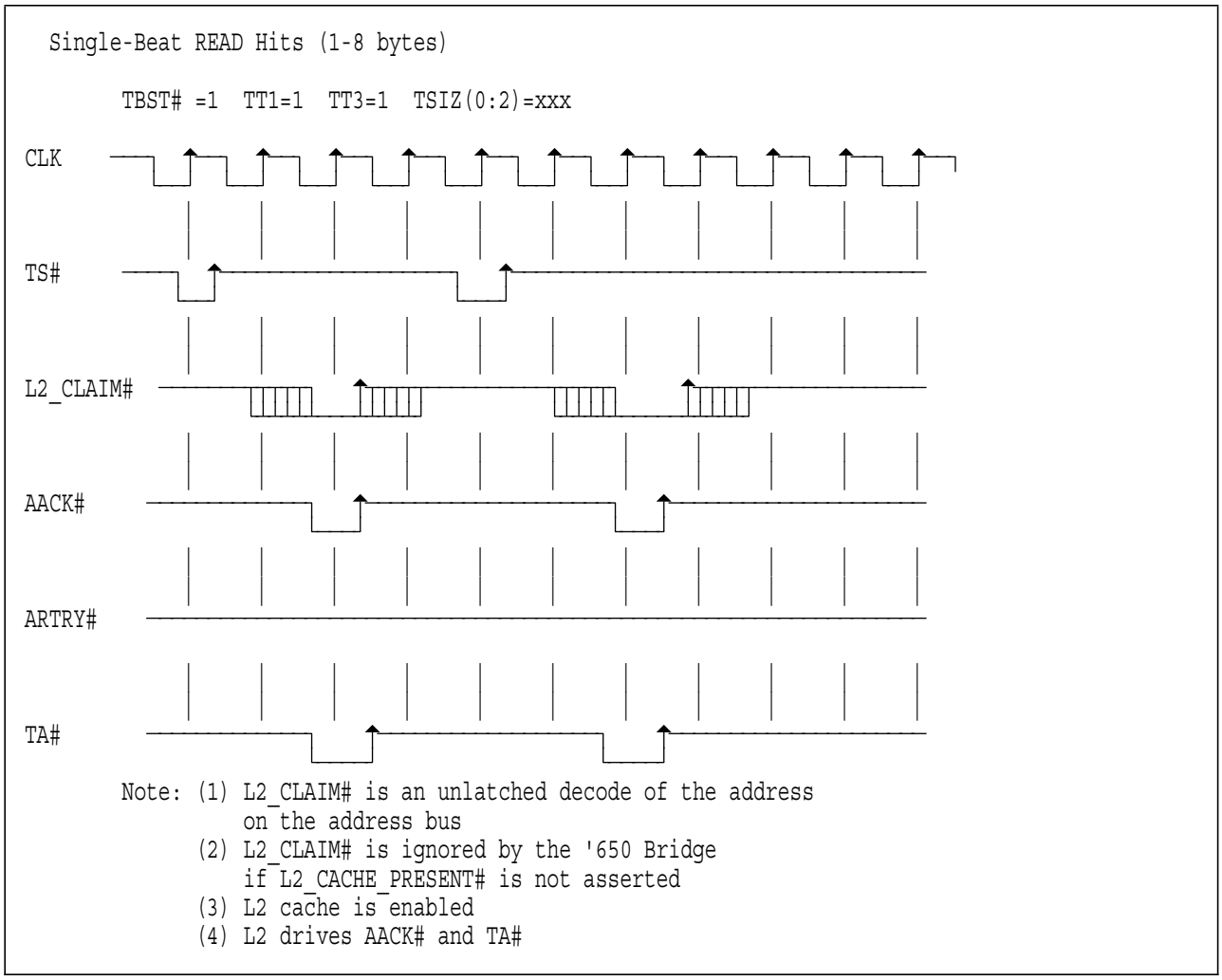


Figure 31. WT L2 Response to Single-Beat READ Hits (1-8 bytes)

Figure 32 shows the WT L2 response for a write cycle. If the cycle is a burst write by the PowerPC processor, the L2 will cache these accesses, overwriting any data that may already be in the cache. Since I/O (PCI) snoop write cycles are never more than 4 bytes, the L2 will invalidate data on write hits, or ignore cycles that are misses.

Since the L2 is a write-through cache design, the L2 never contains “dirty” data. Therefore, the WT L2 will never assert ARTRY# or BR_L2#.

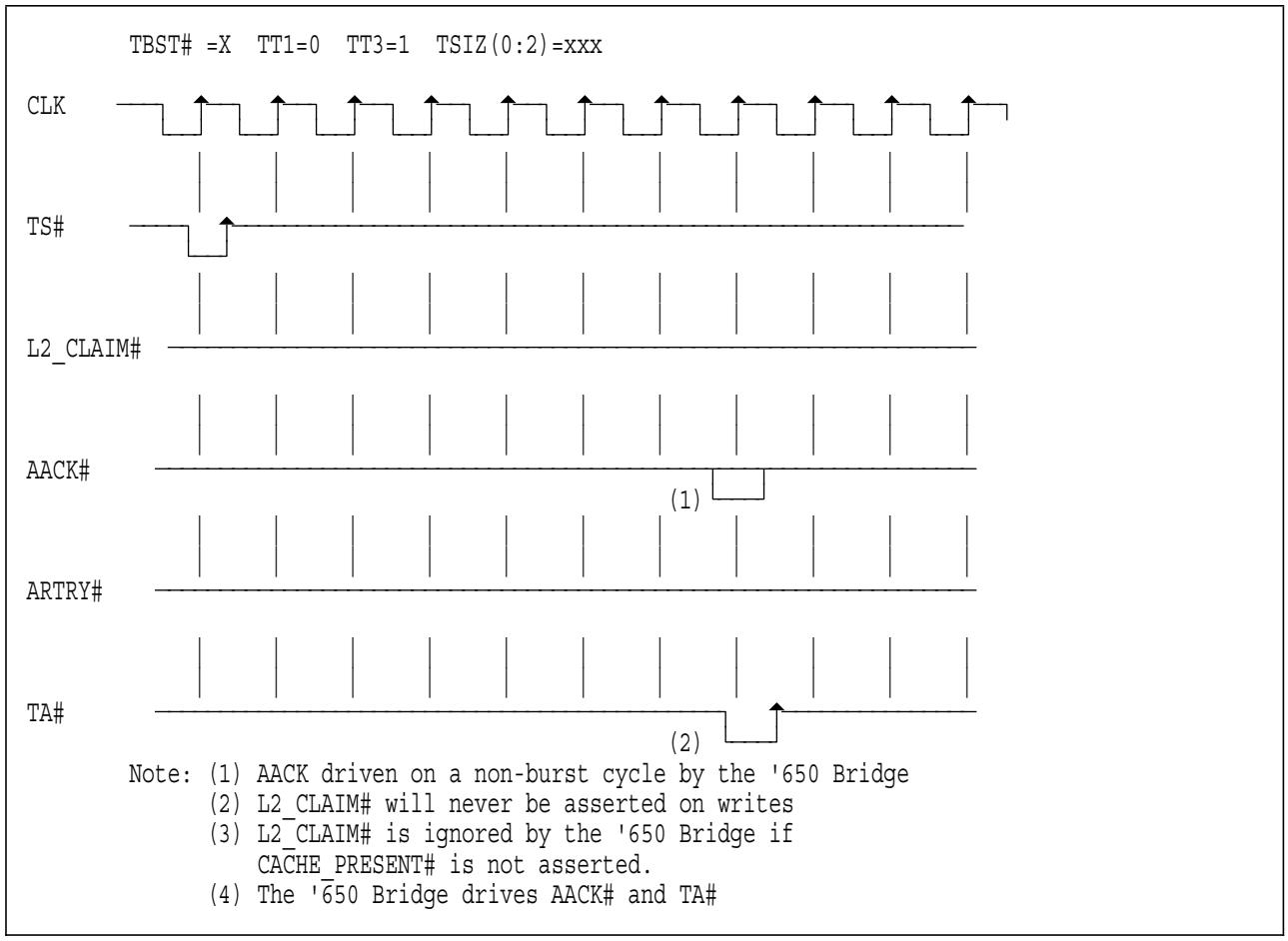


Figure 32. WT L2 Response to Write Cycles

6.7.2.6 Copy-Back L2 (CB L2) Protocol

The CB L2 (also called Write-Back or Store-In) is used to provide the PowerPC processor faster access to memory for both reads and writes. The writes that are cached are from L1 cast-outs. Any writes with kill (due to a snoop hit operation) will not be cached by the CB L2.

Since the CB L2 does cache L1 cast-outs, the performance of these cache cast-outs cycles will be improved (PowerPC processor-to-memory writes).

Coherency is enforced by the MI protocol. That is, data can be in one of three states:

VM Valid Modified

VC Valid Clean

I Invalid

Note: All cachable write operations are “hits” in a CB L2. However, the term “miss” is used to imply that the address of the transfer is to a location not currently stored in the L2, while a hit is to a line marked as valid (either clean or dirty).

The data in the CB L2 is either valid and clean (data is equal to the data in memory), valid and modified (main memory contains stale data), or invalid.

Any cache sector in the CB L2 can be changed from invalid to valid on a burst read or write operation. In order for the data to be stored in the CB L2, the following must be true:

- The cache must be enabled (L2_UPDATE_INH# not asserted)
- CI# must be de-asserted
- L2_MISS_INH# must be de-asserted
- The bus transaction must be a burst read or write that misses the L2
- The bus transaction must not be a write with kill that is a result of a snoop push

On reads, if CI# is asserted and the access is a hit, the CB L2 will ignore the CI# input. If the cycle is a write, and CI# is asserted, the CB L2 will invalidate on a hit, and ignore (not update) on misses.

In addition, if the WT# (write-through) bit is asserted, the CB L2 will cache the data, but NOT assert L2_CLAIM# on writes (entry marked as valid-clean).

Once the data is in the cache, the PowerPC processor can read the data from the cache in any valid byte combinations. Writes of less than 8 bytes that hit in the cache will cause the line to be marked invalid unless the cache includes byte addressability.

6.7.2.6.1 CB L2 READ Responses: The CB L2 maintains coherency in a manner similar to the PowerPC processor.

All processor address bus cycles are snooped whether the address originates with the processor or the ’650 bridge. If any snoop cycle hits the cache, the CB L2 determines whether the snoop hit is caused by the PowerPC processor or the ’650 bridge.

If the read access is from the PowerPC processor, and it hits in the CB L2, then the CB L2 will assert L2_CLAIM# and pace the cycle with AACK and TA.

If the read access is an I/O snoop cycle from the ’650 bridge, the CB L2 will determine if it is a hit. If it is a hit, it will then determine if the data is modified. If the data is not modified, it will do nothing. The CB L2 cache must not assert L2_CLAIM# during any I/O snoop (a snoop originated by the ’650 bridge).

If the data is modified, it will assert ARTRY#. The clock after it asserts ARTRY#, the CB L2 must assert L2_BR# to the ’650 Bridge.

If the PowerPC processor also asserts ARTRY# on the same cycle as the CB L2, the ’650 Bridge will grant the bus to the processor. The CB L2 must sample the 60X_BR# on the clock after ARTRY#. If the 60X_BR# is active, the CB L2 must de-assert its bus request on the following clock. (The CB L2 will update its data when the PowerPC processor pushes its data and therefore the data in the L2 is superfluous.)

If the L2_BR# was already active before the snoop cycle, and the current snoop cycle does not require the CB L2 to write to memory, the CB L2 must de-assert its bus request the cycle after ARTRY#. The CB L2 can then re-assert its bus request[1].

6.7.2.6.2 CB L2 WRITE Responses: The CB L2 must fully decode the TT bits and the CI# and WT# bits. On any write misses that are marked as either write through, cache inhibited, or writes with kill, the CB L2 will not assert L2_CLAIM#.

The CB L2 will not update its cache on write misses of less than 32 bytes (TBST# must be active).

---

[1] Note that, in an implementation which is a single-processor system, the arbiter does not need to see only the one BR# active from the processor. As mentioned earlier, if both the L2_BR# and the 60X_BR# are active, the arbiter will always grant the next cycle to the PowerPC processor. The L2_BR# will then be sampled after the processor cycle is complete to determine if it also needs the bus.

The CB L2 should buffer any data that is written or read by the PowerPC processor that is to be cached by the L2 but requires a cache push operation by the L2 to precede its writing into the cache. (This situation happens when the processor reads or writes a cache line in the L2, there is a miss, and the data stored at that tag location is dirty.) This buffer can be of variable depth. If the buffer is full (from previous operation(s)), and the current cycle is cachable but requires a push, the CB L2 MUST not assert L2_CLAIM# or ARTRY#. Instead, the CB L2 should just not cache these cycles. This is required in order to maintain the greatest bandwidth from memory to the PowerPC processor and vice versa[2].

6.7.2.6.2.1 PowerPC Processor Writes: The CB L2 cache will cache data on the following write transactions:

- A burst write that hits a clean line
- A burst write that hits a dirty line
- A burst write that misses a line (only if the buffer is not full)
- A non-burst write that hits a clean line
- A non-burst write that hits a dirty line
- A non-burst write that misses a line (only if the buffer is not full)

On a burst write that is a hit to the same address that is already cached, the CB L2 will assert L2_CLAIM# and overwrite the previous data (whether modified or not).

If the address is a hit that is to a different address than that in the L2 cache, and the currently cached data is clean, the cycle completes as before.

If the cycle is not a burst, then the CB L2 will determine whether or not the cycle hits in the cache. If the cycle hits in the cache to either a clean or a dirty sector of the same address, the CB L2 will assert L2_CLAIM# and update the required byte(s). If the cycle would overwrite dirty data to another address, the CB L2 will NOT assert L2_CLAIM#. That is, the CB L2 cannot have any entry of less than 32 bytes cached.

If the write is a miss and the buffer is full, the CB L2 may do either of the following:

- Assert ARTRY# and BR_L2#
- Nothing (preferred)

6.7.2.6.2.2 I/O Snoop Writes: All processor address bus cycles are snooped whether the address originates with the processor or the ’650 bridge. The CB L2 determines whether the snoop is caused by the PowerPC processor or the ’650 bridge. The CB L2 must not assert L2_CLAIM# on any I/O snoop.

If the write access is an I/O snoop cycle from the ’650 bridge, the CB L2 will determine if it is a hit. If it is a miss, it will do nothing. If it is a hit, the CB L2 will then determine if the data is modified. If the data is not modified, it will will mark that line as invalid.

If the data is modified, it will assert ARTRY#. The clock after it asserts ARTRY#, the CB L2 must assert L2_BR# to the ’650 Bridge.

If the PowerPC processor also asserts ARTRY# on the same cycle as the CB L2, the ’650 Bridge will grant the bus to the processor. The CB L2 must sample the 60X_BR# on the clock after ARTRY#. If the 60X_BR# is active, the CB L2 must de-assert its bus request on the following clock.

---

[2] The reason for not asserting ARTRY# on a CB L2 pipeline (buffer) full condition is that the PowerPC processor will be delayed by at least 20 cycles since the L2 would need to write out the data, and then the processor would need to re-read (write) the data from (to) memory.

If the L2_BR# was already active before the snoop cycle, and the current snoop cycle does not require the CB L2 to write to memory, the CB L2 must de-assert its bus request the cycle after ARTRY#. The CB L2 can then re-assert its bus request[3].

6.7.2.7 CB L2 Timing Diagrams

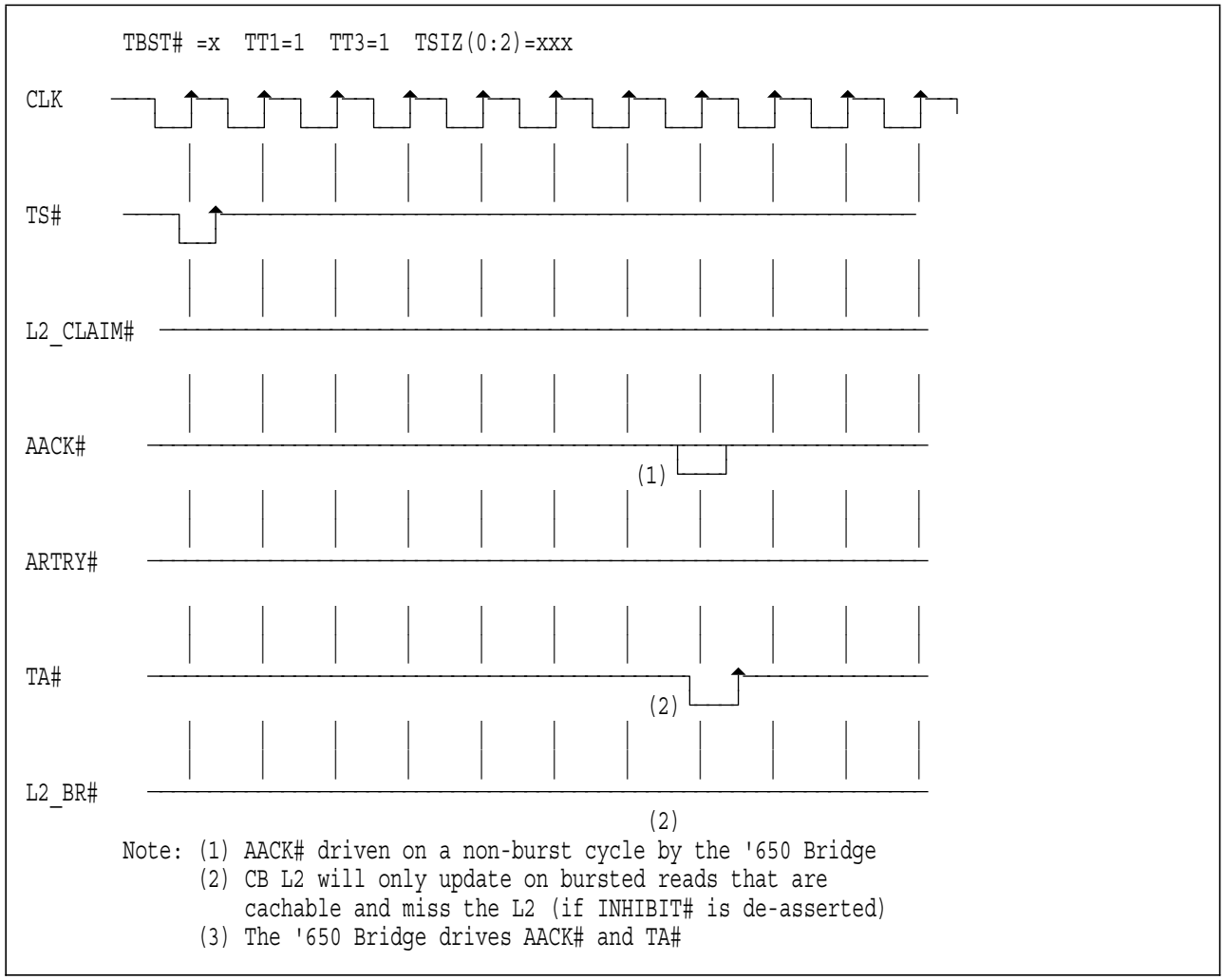


Figure 33. CB L2 Response to Read Misses

[3] Note that, in an implementation which is a single-processor system, the arbiter does not need to see only the one BR# active from the processor. As mentioned earlier, if both the L2_BR# and the 60X_BR# are active, the arbiter will always grant the next cycle to the PowerPC processor. The L2_BR# will then be sampled after the processor cycle is complete to determine if it also needs the bus.

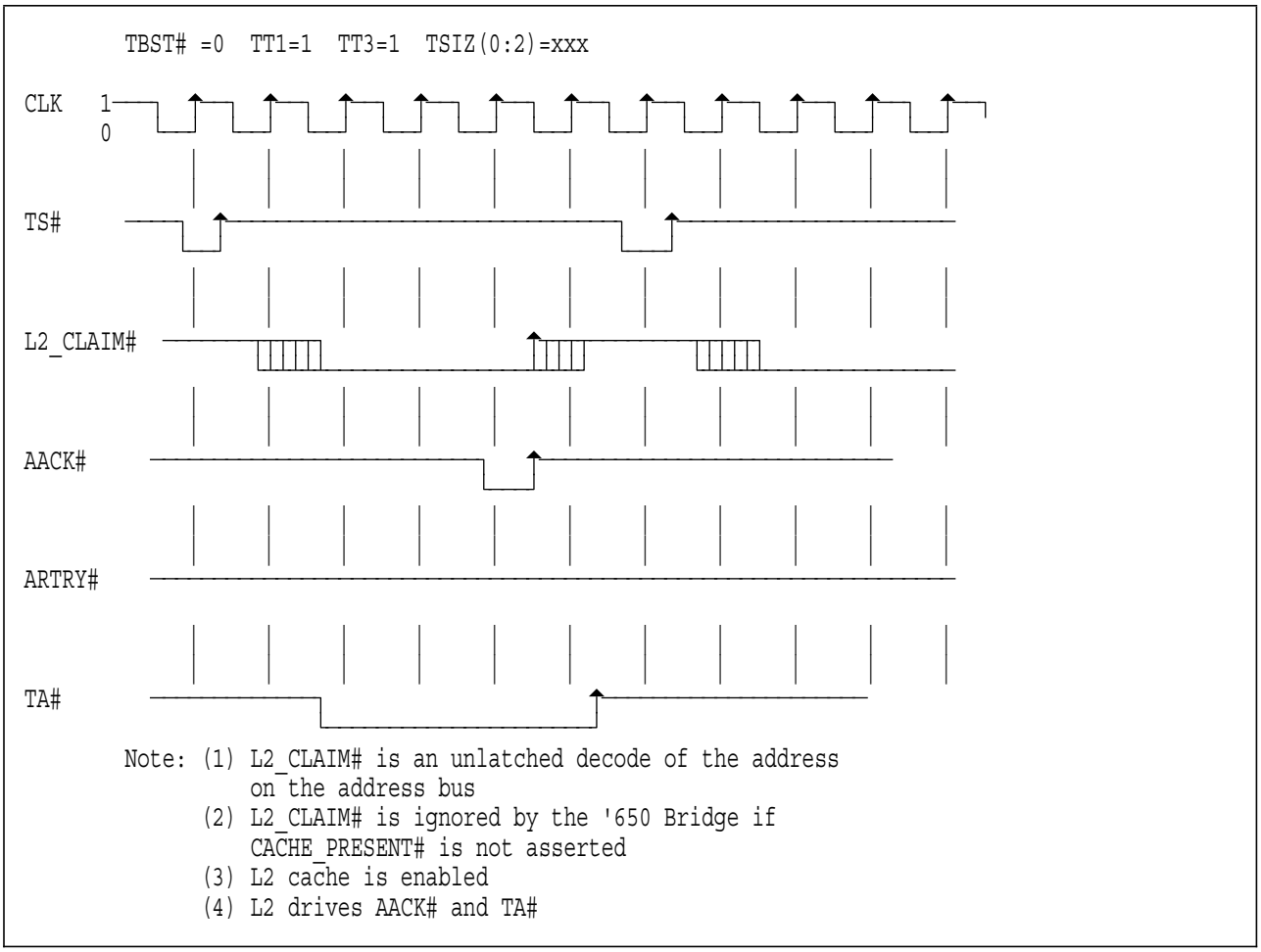


Figure 34. CB L2 Response to Burst READ Hits

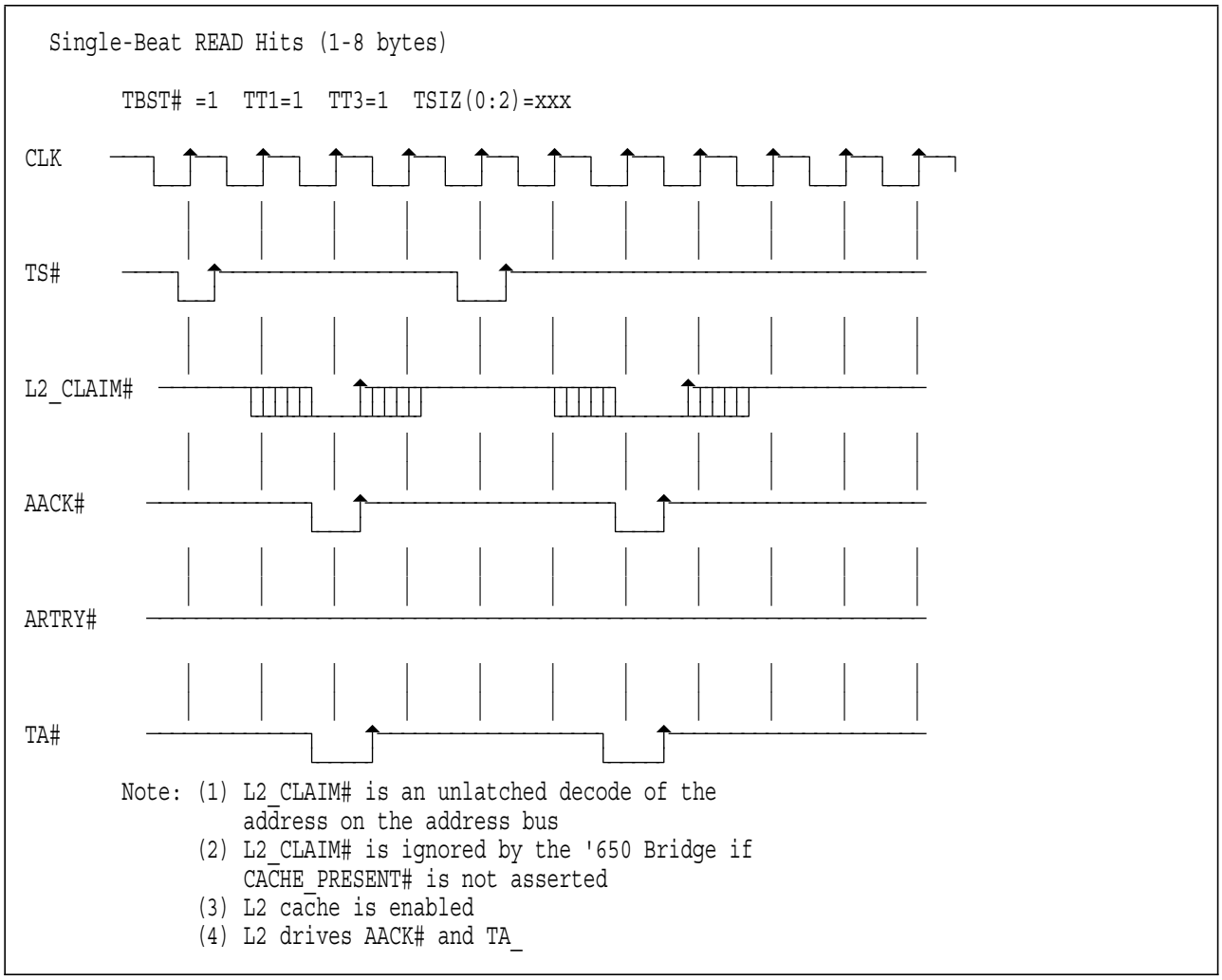


Figure 35. CB L2 Response to Single-Beat READ Hits (1-8 bytes)

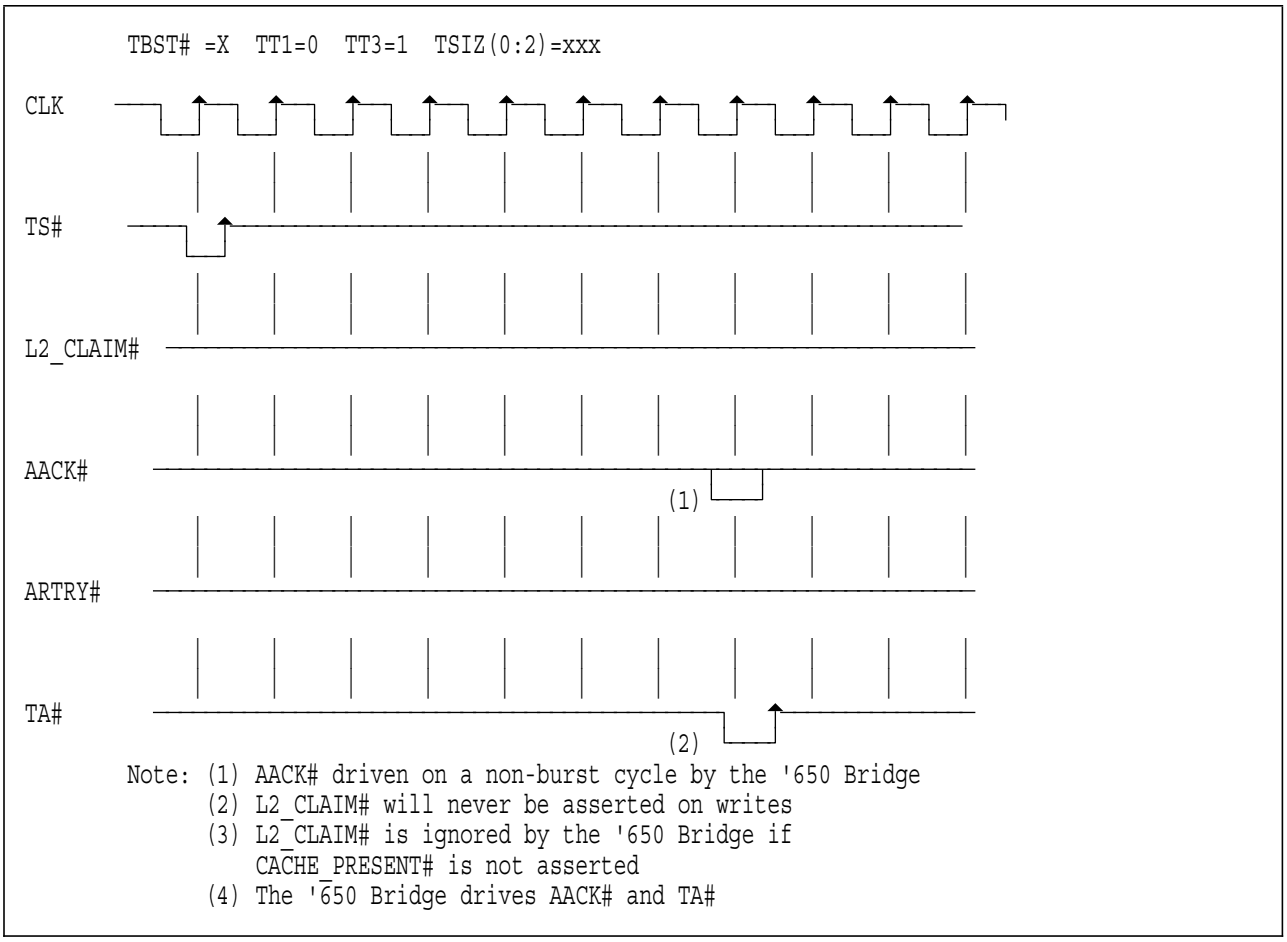


Figure 36. CB L2 Response to Write Cycles

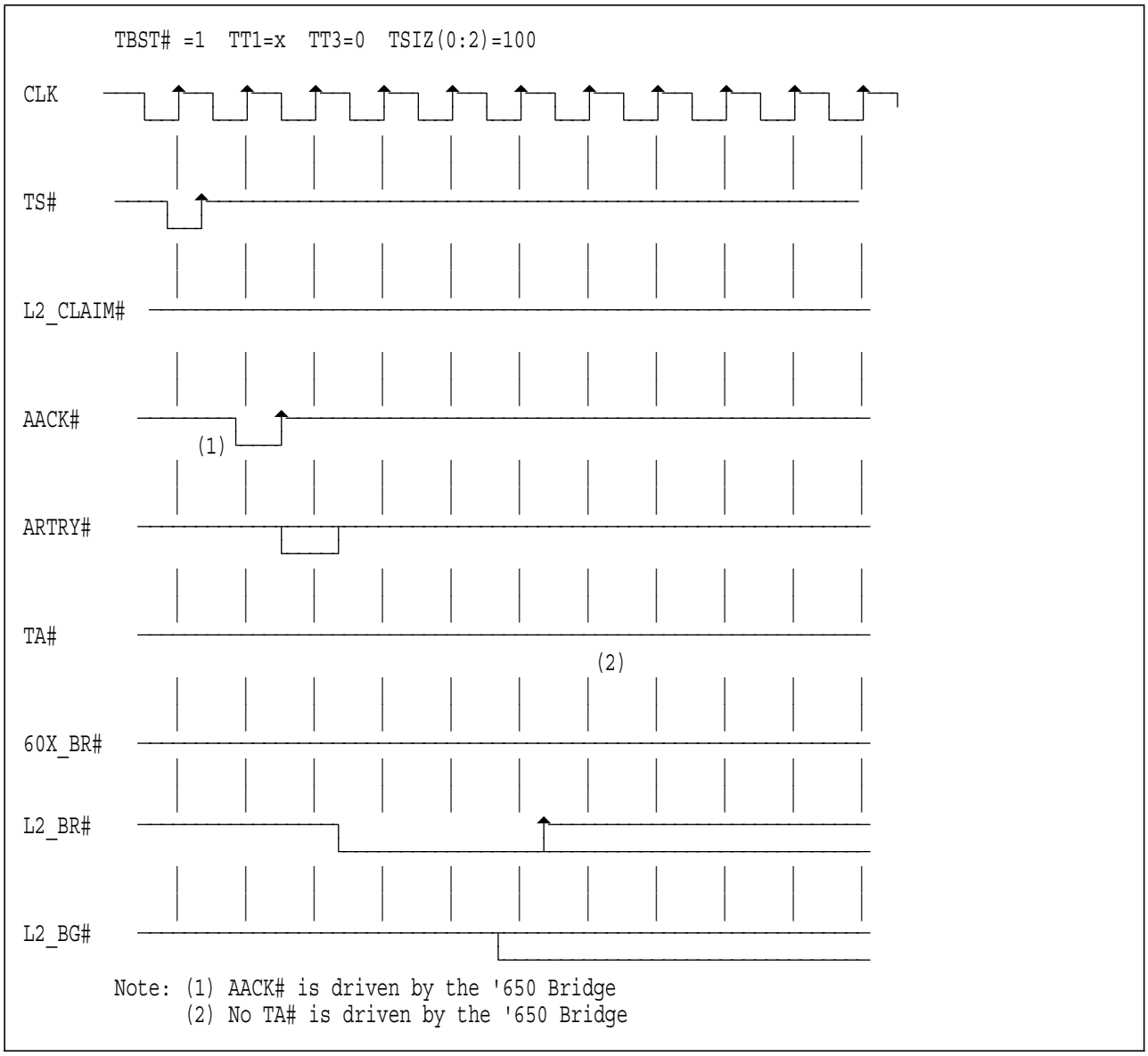


Figure 37. WT L2 Response to I/O Snoop Hits without PowerPC Processor Snoop Hit

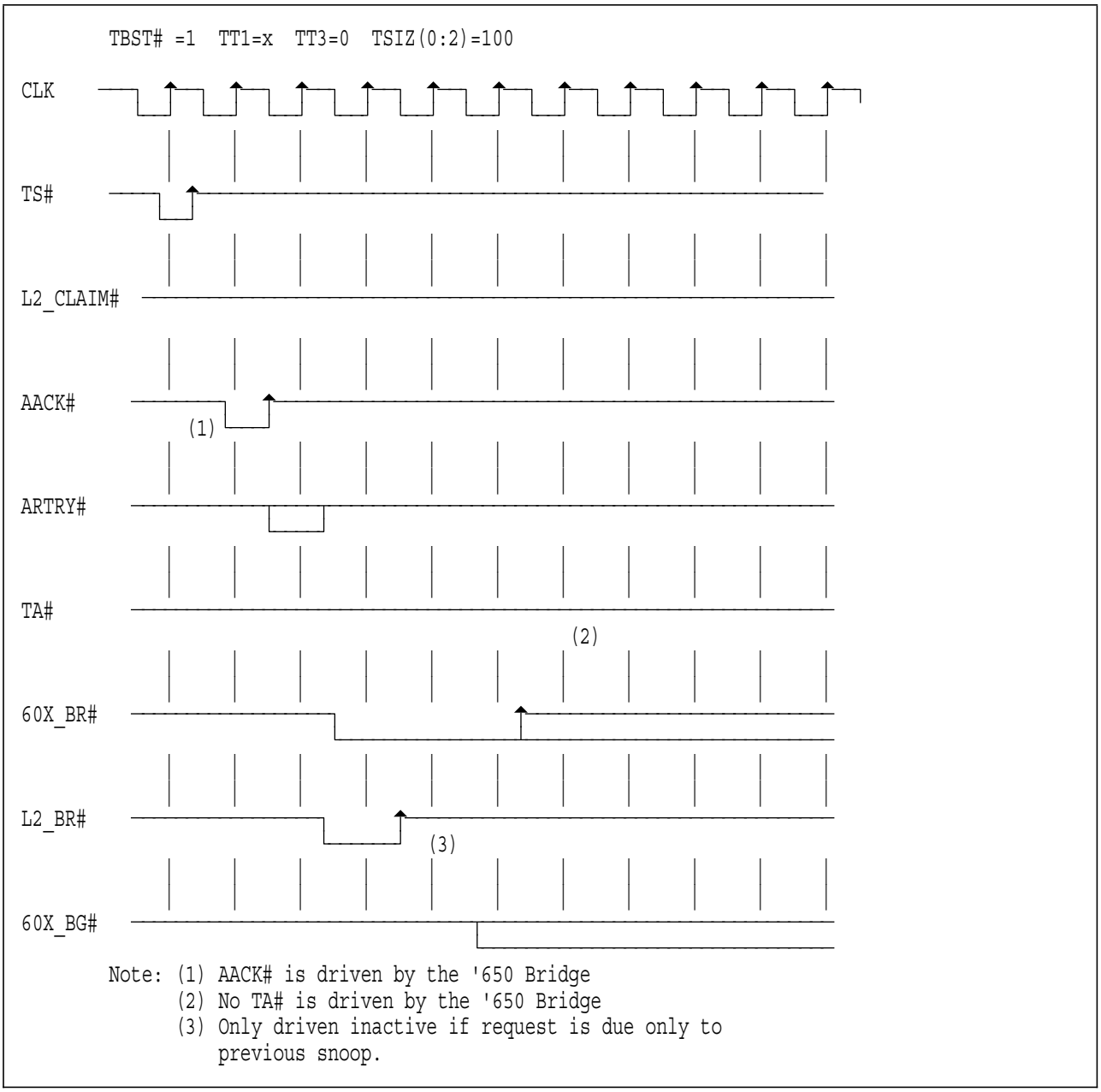
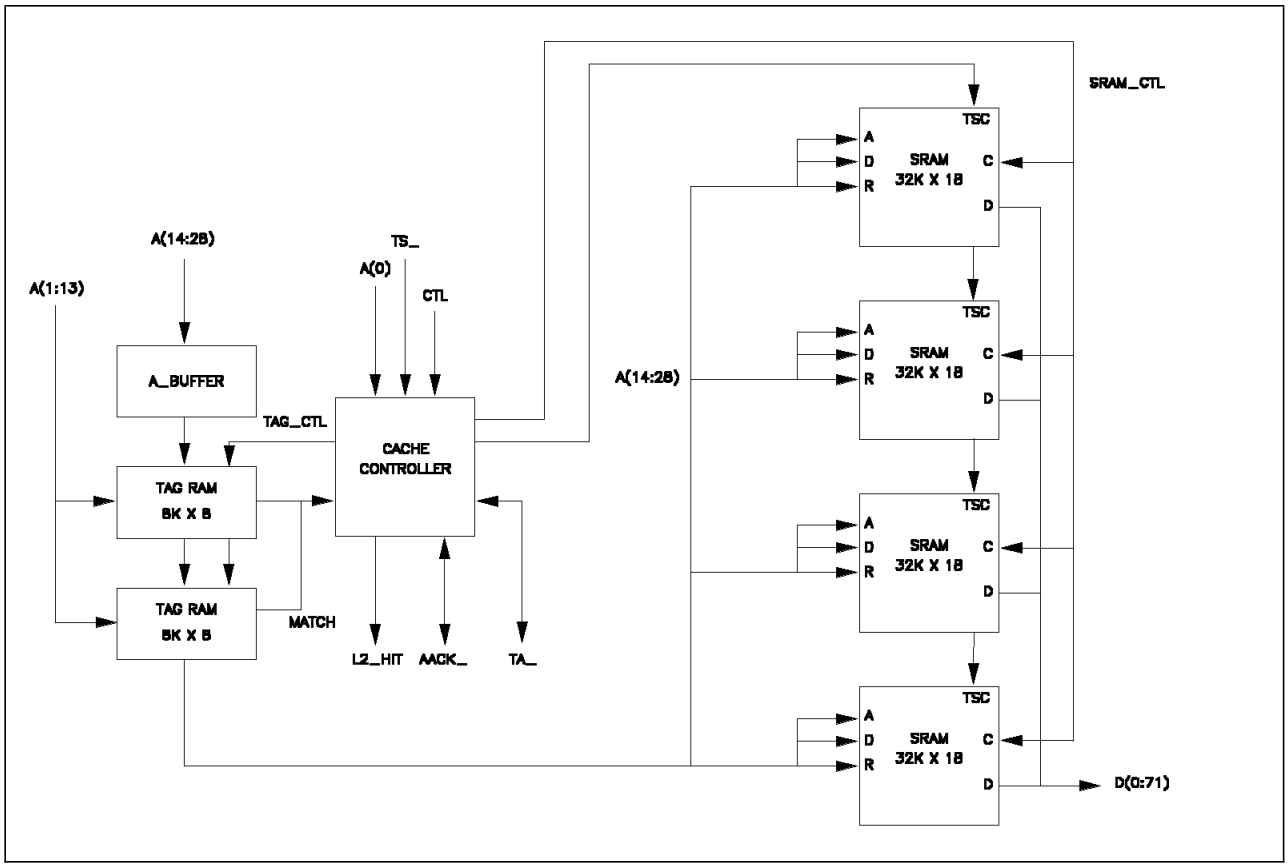


Figure 38. CB L2 Response to I/O Snoop Hits with PowerPC Processor Snoop Hit

6.7.3 L2 Cache Design

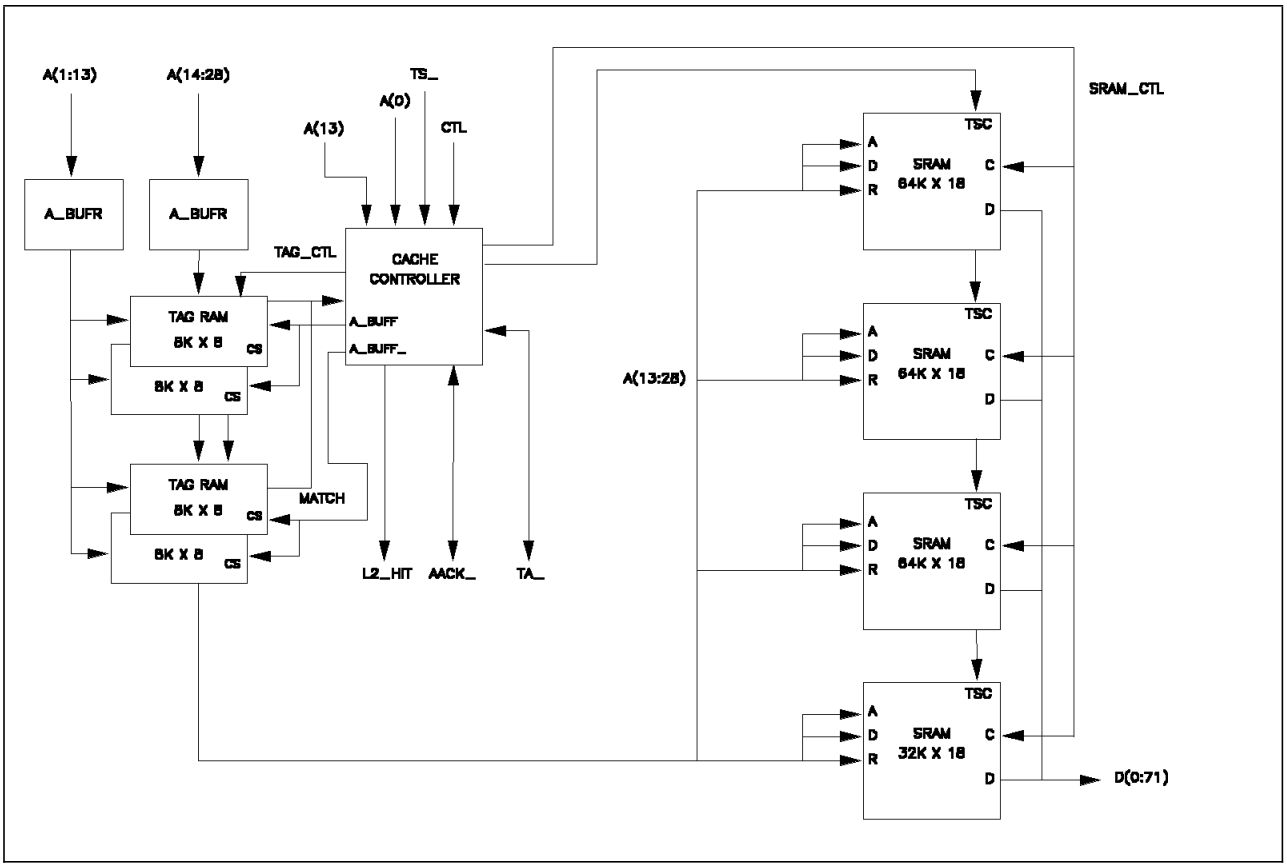
This section defines the design of an L2 Cache. Figure 39 gives a block diagram of the structure for a / 256-KB cache, and Figure 40 gives a block diagram for a 512-KB cache. For both designs, the cache con- / troller may be an IBM 8184995, and the Tag RAM may be an IDT 71B74 or equivalent. For the 256-KB / cache, the SRAMs may be Motorola MCM67M518, which is a 32Kx18 BurstRam, or an equivalent / product. The SRAMs for the 512-KB cache may be Motorola MCM67M618, which is a 64Kx18 / BurstRam, or an equivalent product.



/ Figure 39. 256-KB L2 Block Diagram

The control signals for this cache controller are as follows:

Signals	*Description*
VCC (4 pins)	POWER. Plus-5-volt power to controller.
GND (4 pins)	POWER. GND to controller.
A_IN	INPUT. Address In. Address input for Tag RAM bank select (used only for 512-KB L2).
A_EN	INPUT. Address Buffer Enable. Active (high) enables A_BUFF and A_BUFF_. Inactive (low) disables A_BUFF (low) and A_BUFF_ (high).
A_BUFF	OUTPUT. Address Buffer Output. Non-inverted buffer of A_IN when A_EN is active (high). Low when A_EN is inactive (low).
A_BUFF_	OUTPUT. Address Buffer Output. Inverted buffer of A_IN when A_EN is active (high). High when A_EN is inactive (low).
ARTRY_HIT_	OUTPUT. L2 Hit with ARTRY_ timings. This signal is a synchronous hit signal which goes active (low) after clock 2. The signal can be sampled by the system on clock 3. This signal is valid for one clock cycle only.
OE_	INPUT. When active (low), all outputs of the controller will be driven. When inactive (high), all outputs are tri-stated.
BUS_CLOCK0	INPUT. Bus Clock. Card I/O. This clock drives all state machine changes. All changes occur on the rising edge of this clock.



/ Figure 40. 512-KB L2 Block Diagram

HRESET_ INPUT. Reset. When active (low), this signal resets the controller to its initial states and will generate the control signals to flush the cache Tag RAM.

A0 INPUT. Card I/O. This MSb of the address bus determines if the address is below 2 GB. Used as a qualifier in determining if the L2 will respond to the address presented on the PowerPC processor bus.

L2_CACHE_FLUSH_ INPUT. Card I/O. Same as the card.

L2_CACHE_DIS_ INPUT. Card I/O. Same as the card.

L2_CACHE_INH_ INPUT. Card I/O. Same as the card.

CI_ INPUT. Card I/O. Same as the card.

MATCH INPUT. Active (high) signal from the cache Tag RAM indicates that the address presented on the PowerPC processor bus has been cached in the L2. (This is a “hit.”) Inactive (low) signal indicates that the address is not in the L2 cache. (This is a “miss.”)

L2_CACHE_HIT_ OUTPUT. Card I/O. Same as the card.

TA_ INPUT. PowerPC Processor Transfer Acknowledge. Used to determine when data placed on the data bus is valid.

TA_OUT_ OUTPUT. Active (low) during cache reads to indicate that valid data has been placed on the PowerPC processor bus. This signal MUST be buffered external to the controller with an open-collector (or tri-statable) buffer.

AACK_ INPUT. PowerPC Processor Address Acknowledge. Active during reads of the cache to signal the system that a new address can be placed on the bus. Tri-stated when not active.

TSIZ(2..0) INPUT. PowerPC Processor Transfer Size. Card I/O. Same as the card.

TBST_ INPUT. PowerPC Processor Transfer Burst. Card I/O. Same as the card.

TS_ INPUT. PowerPC Processor Transfer Start. Card I/O. Same as the card.

TT(4..0) INPUT. PowerPC Processor Transfer Type. Card I/O. Same as the card.

TAG_VALID OUTPUT. Is active (high) when entry for Tag RAM is valid. Inactive (low) during a Tag RAM write will invalidate the tag. During a read of the Tag RAM, an active (high) will force the Tag RAM to match only validated memory locations.

TAG_WE_ OUTPUT. Writes the data on the inputs to the Tag RAM when the signal transitions from active (low) to inactive (high).

TAG_RESET_ OUTPUT. When active (low), all tags in the Tag RAM will be reset and invalidated.

SRAM_TSC_ OUTPUT. When active (low), the Burst SRAMs latch address.

SRAM_S1_ OUTPUT. When active (low), the Burst SRAMs are enabled.

SRAM_BAA_ OUTPUT. When active (low), the address in the Burst SRAMs is incremented on the next rising clock edge.

SRAM_W_ OUTPUT. When active (low), the data on the inputs to the Burst SRAMs will be stored at the location pointed to by the address on the next rising edge of the clock.

SRAM_G_ OUTPUT. When active (low), data will be driven by the Burst SRAM on the data lines.

6.7.4 Upgrade Slot Pin Definition

This section defines the pins for the upgrade slot.Figure 41 shows a diagram of this 2x101 Micro Channel-style connector. It consists of two rows of 101 pins. The 101 pins include the 10 pins from VA10 through VA1, 49 pins from A1 through A49, and 42 pins from A52 through A93.

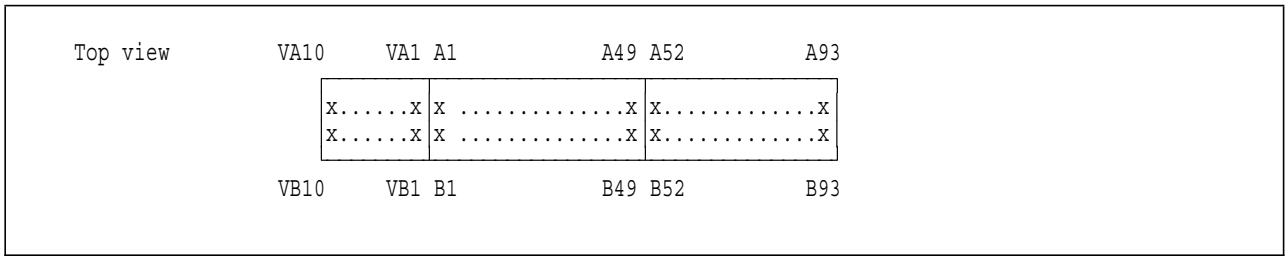


Figure 41. Upgrade/L2 Cache Connector

Table 23 (Page 1 of 4). Upgrade Processor/L2 Cache Connector Pin Assignments

Pin	Function	Pin	Function
A1	L2_CLAIM#	B1	L2_TAG_CLR#
A2	60X_A(9)	B2	DATA_BUS_GNT#, BG_60X#
A3	GROUND	B3	60X_A(10)
A4	60X_A(11)	B4	60X_A(12)

Table 23 (Page 2 of 4). Upgrade Processor/L2 Cache Connector Pin Assignments			
Pin	**Function**	**Pin**	**Function**
A5	60X_A(13)	B5	GROUND
A6	60X_A(14)	B6	60X_A(15)
A7	+5 VOLTS	B7	60X_A(16)
A8	60X_A(18)	B8	60X_A(17)
A9	60X_A(19)	B9	+5 VOLTS
A10	60X_A(0)	B10	60X_A(20)
A11	GROUND	B11	60X_A(1)
A12	60X_A(2)	B12	60X_A(3)
A13	60X_A(4)	B13	GROUND
A14	60X_A(5)	B14	60X_A(6)
A15	+5 VOLTS	B15	60X_A(7)
A16	60X_A(21)	B16	60X_A(8)
A17	60X_A(22)	B17	+5 VOLTS
A18	60X_A(23)	B18	60X_A(24)
A19	GROUND	B19	60X_A(25)
A20	60X_A(26)	B20	60X_A(27)
A21	60X_A(28)	B21	GROUND
A22	60X_A(29)	B22	60X_A(30)
A23	+5 VOLTS	B23	60X_A(31)
/ A24	GROUND	B24	GROUND
/ A25	TRANSFER TYPE (4), TT(4)	B25	GROUND
A26	TRANSFER TYPE (2), TT(2)	B26	TRANSFER TYPE (3), TT(3)
A27	TRANSFER TYPE (0), TT(0)	B27	TRANSFER TYPE (1), TT(1)
A28	TRANSFER SIZE (1), TSIZ(1)	B28	TRANSFER SIZE (2), TSIZ(2)
A29	TRANSFER_BURST#, TBST#	B29	TRANSFER SIZE (0), TSIZ(0)
A30	TRANSFER_ACKNOWLEDGE#, TA#	B30	GROUND
A31	GROUND	B31	ADDRESS_ACKNOWLEDGE#, AACK#
A32	DATA_RETRY#, DRTRY#	B32	ADDRESS_RETRY#, ARTRY#
A33	TRANSFER_ERR_ACK#, TEA#	B33	EXT_ADDR_TRANS_START#, XATS#
A34	GROUND	B34	GROUND
A35	BUS_REQUEST_60X#, BR_60X#	B35	DATA_PARITY_ERROR#, DPE#
A36	BUS_REQUEST_L2#, BR_L2#	B36	BUS_GRANT_60X#, BG_60X#
A37	+5 VOLTS	B37	BUS_GRANT_L2#, BG_L2#
A38	TRANSFER_START#, TS#	B38	60X_INTERRUPT#, INT_60X
A39	GROUND	B39	+5 VOLTS
/ A40	GROUND	B40	GROUND
A41	GROUND	B41	BUS_CLOCK_0
A42	BUS_CLOCK_1	B42	GROUND
A43	GROUND	B43	BUS_CLOCK_2
A44	60X_D(31)	B44	+5 VOLTS
/ A45	60X_D(30)	B45	GROUND
A46	60X_D(28)	B46	60X_D(29)
A47	GROUND	B47	60X_D(27)
A48	60X_D(25)	B48	60X_D(26)

Table 23 (Page 3 of 4). Upgrade Processor/L2 Cache Connector Pin Assignments			
Pin	**Function**	**Pin**	**Function**
A49	60X_D(24)	B49	GROUND
A50	<key>	B50	<key>
A51	<key>	B51	<key>
A52	60X_D(22)	B52	60X_D(23)
A53	+5 VOLTS	B53	60X_D(21)
A54	60X_D(19)	B54	60X_D(20)
A55	60X_D(18)	B55	+5 VOLTS
A56	60X_D(16)	B56	60X_D(17)
A57	GROUND	B57	60X_D(15)
A58	60X_D(13)	B58	60X_D(14)
A59	60X_D(12)	B59	GROUND
A60	60X_D(10)	B60	60X_D(11)
A61	GROUND	B61	60X_D(9)
A62	60X_D(7)	B62	60X_D(8)
A63	60X_D(6)	B63	+5 VOLTS
A64	60X_D(5)	B64	60X_D(4)
A65	+5 VOLTS	B65	60X_D(3)
A66	60X_D(1)	B66	60X_D(2)
A67	60X_D(0)	B67	GROUND
A68	60X_D(62)	B68	60X_D(63)
A69	60X_D(61)	B69	+5 VOLTS
A70	60X_D(59)	B70	60X_D(60)
A71	GROUND	B71	60X_D(58)
A72	60X_D(56)	B72	60X_D(57)
A73	60X_D(55)	B73	GROUND
A74	60X_D(53)	B74	60X_D(54)
A75	+5 VOLTS	B75	60X_D(52)
A76	60X_D(50)	B76	60X_D(51)
A77	60X_D(49)	B77	GROUND
A78	60X_D(47)	B78	60X_D(48)
A79	GROUND	B79	60X_D(46)
A80	60X_D(44)	B80	60X_D(45)
A81	60X_D(43)	B81	+5 VOLTS
A82	60X_D(42)	B82	60X_D(41)
A83	+5 VOLTS	B83	60X_D(40)
A84	60X_D(38)	B84	60X_D(39)
A85	60X_D(37)	B85	GROUND
A86	60X_D(35)	B86	60X_D(36)
A87	+5 VOLTS	B87	60X_D(34)
A88	60X_D(32)	B88	60X_D(33)
A89	GROUND	B89	+5 VOLTS
A90	60X_DATA_PARITY_(6)	B90	60X_DATA_PARITY_(7)
A91	60X_DATA_PARITY_(4)	B91	60X_DATA_PARITY_(5)
A92	60X_DATA_PARITY_(2)	B92	60X_DATA_PARITY_(3)

Table 23 (Page 4 of 4). Upgrade Processor/L2 Cache Connector Pin Assignments			
Pin	**Function**	**Pin**	**Function**
A93	60X_DATA_PARITY_(0)	B93	60X_DATA_PARITY_(1)
VA1	GROUND	VB1	FULL_SPEED
VA2	+5 VOLTS	VB2	60X_CACHE_INHIBIT#, CI#
VA3	60X_WRITE_THROUGH#, WT#	VB3	60X_GLOBAL#, GBL#
VA4	60X_SHARED#, SHD#	VB4	BCLK_SPEED1
VA5	BCLK_SPEED0	VB5	HARD_RESET#, HRESET#
VA6	SOFT_RESET#, SRESET#	VB6	UPGRADE_PROCESSOR_PRESENT#
VA7	L2_CACHE_PRESENT#	VB7	+5 VOLTS
VA8	L2_CACHE_256K	VB8	L2_CACHE_CB (COPY-BACK)
VA9	L2_UPDATE_INH#	VB9	L2_MISS_INH_#
VA10	TRANSFER_CODE_(1), TC(1)	VB10	TRANSFER_CODE_(0), TC(0)

Appendix A. Implementation Examples

The following subsections give diagrams and brief descriptions of example implementations for a portable, an energy-managed workstation, a medialess workstation, a technical workstation, a server, and a symmetric multiprocessor. A description of a diagnostic strategy is also included.

A.1 Portable

This appendix gives a basic hardware overview of a PowerPC Reference Platform-compliant portable system. Figure 42 shows a schematic diagram of the components of this portable system. This portable configuration uses the PowerPC 603 processor, and provides one serial port, one parallel port, various audio jacks, built-in microphone and speakers, one keyboard/mouse port, one video graphics port, one SCSI-2 port, and two type-2 PCMCIA sockets.

A.1.1 Subsystems

Major subsystems are connected to the internal +5.0-volt, 32-bit PCI bus. A memory controller, PCI-ISA Bridge, SCSI controller, power management controller, and video graphics controllers are located on this bus.

Native and extended I/O controllers, Real-Time Clock, business audio, and PCMCIA are located on the internal ISA bus.

A.1.2 PowerPC 603 Processor

The PowerPC 603 processor is configured to have 32 address bits and and can be configured to have 32 data bits. Internally, the 603 has an 8-KB instruction cache and an 8-KB data cache.

The 603 processor has a number of differences from the 601. Please consult the 603 documentation for details. Two changes in particular should be noted: the 603 does not snoop the internal instruction cache, and the 603 Endian switching instructions are different than the 601 instructions.

The following process will switch the system from Big-Endian to Little-Endian mode when used in the Reference Implementation with a PowerPC processor other than the PowerPC 601 microprocessor. Because this process must execute during transition periods when the processor and system components are in different Endian modes, care must be taken to assure that interrupts are not taken or that data and instructions do not remain in cache in the wrong Endian order. The instructions to perform this switch must not span a page boundary. The process is outlined below:

a) Enable address translation
b) Flush all system caches
c) Disable interrupts
d) Enter supervisor state
e) Invalidate the instruction cache
f) Set the processor and system state to Little-Endian (see Figure 43)

   Note: Processor and system are now in Little-Endian mode. All instructions must be in Little-Endian order.

g) Put interrupt handlers and processor data structures in Little-Endian format
h) Enable interrupts
i) Start the Little-Endian operating system initialization

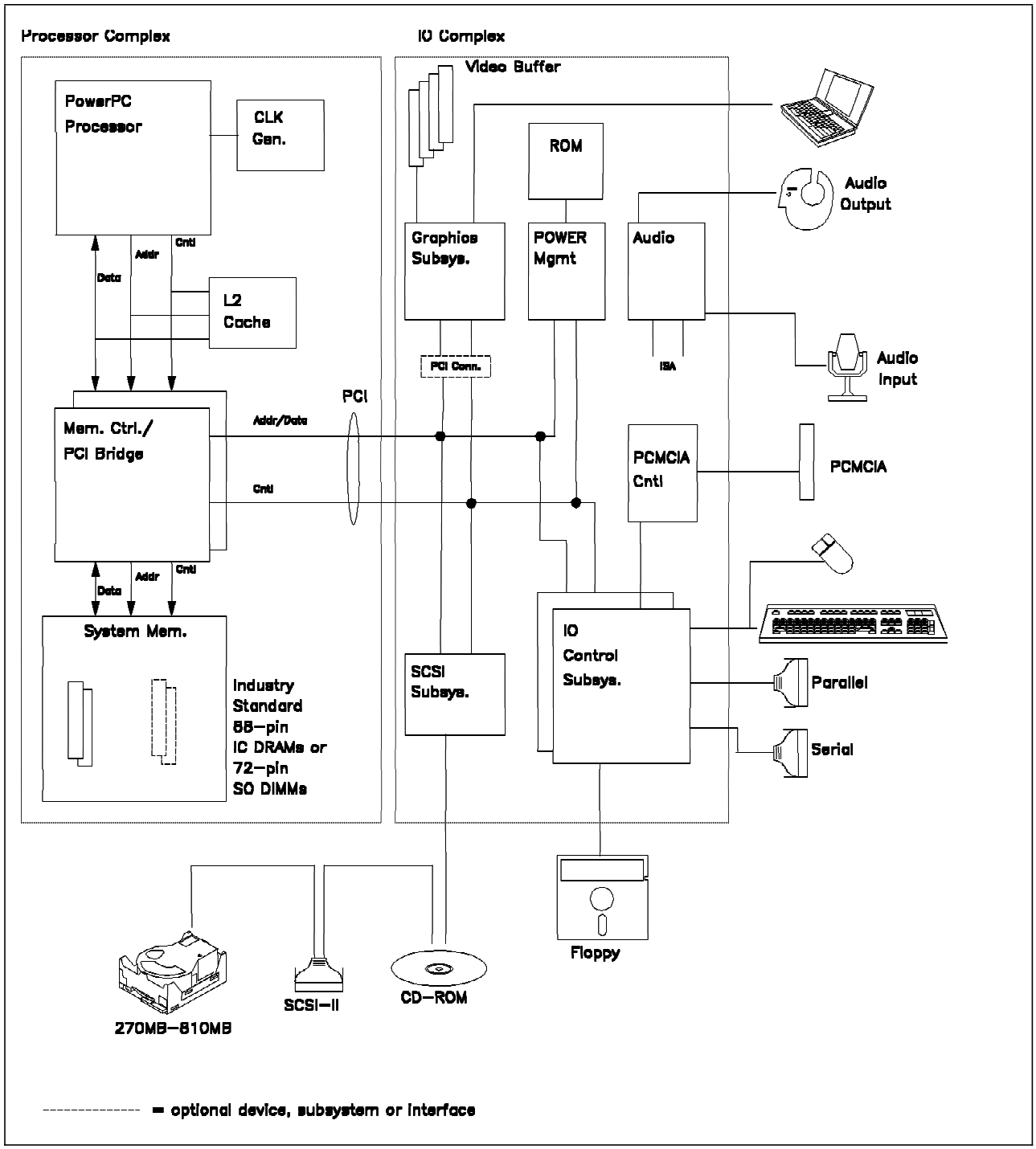


Figure 42. PowerPC Reference Platform Recommended Portable System

```
/   ;Set the MSR ILE bit (bit 15) here.  It is not copied from SRR1.
       x00  mfmsr     R9   ;Load current MSR register
       x04  ori  R9,R9,MSR_LE    ;Set the LE bit in the MSR (bit 31)
       x08  mtsrr1    R9   ;Store the target MSR in SRR1
/      xxC  bl   x    ;Address of next instruction in Link Register
/   x:
/      x10  mfspr R2,8      ;Link register to R2
/      x14  addi  R2,R2,y-x  ;Calculate instruction after RFI
/      x18  mtsrr0    R2   ;Set branch address in SRR0
       x1C  rfi       ;Return from interrupt to set LE mode
                 ;The RFI performs the necessary synchronization actions
    y:
       x20  stb  R5,0(R29)  ;Set system components to LE
/   ;Register R5 has the data and R29 has the address for the Endian control port
/   ;Endian control port at X'8000 0092' must be addressed at
/   ;X'8000 0095' because processor is modifying addresses now
    ;To this point all instructions are in Big-Endian Format
/   ;Include a string of palindromic instructions to pass time until the system
/   ;completes the switch.  Twenty five are suggested based on a 66 MHz processor
/      x24  addi  R0,R1,0x138     ;Instructions which work LE or BE
/                       ;This instruction generates 0x38010138
/      x28  addi  R0,R1,0x138
/      x2C  addi  R0,R1,0x138
/      ...
/      xyy  addi  R0,R1,0x138
/   ;Start of Little Endian instructions
```

Figure 43. PowerPC Instruction Stream to Switch Endian Modes

/ This switching process is only applicable for PowerPC processors in a design with a Bi-Endian memory as / shown in Figure 56.

/ If the design uses the Big-Endian memory approach as shown in Figure 57, then this process must be modi- / fied. The Little-Endian portion of the operating system loader and any of the rest of the operating system / which was brought into memory before the system was switched to Little-Endian mode does not reside in / memory in the correct format. This information was brought into memory in the true Little-Endian format / as it existed on the media. It must exist in memory in the “PowerPC Little-Endian” form. This form has / the bytes reversed within each doubleword. After the system is in Little-Endian mode, further input from / media of Little-Endian information will arrive in system memory in the correct form. Examples of / approaches to accomplish this byte reversal are as follows:

/ • Before the processor switches to Little-Endian mode, load and store each Little-Endian doubleword, / doing a byte reversal in the process.
/ • Swap out the Little-Endian material while in the Big-Endian mode and swap it back in after the system / is in Little-Endian mode and before the processor has been switched to Little-Endian mode. The / instructions to perform the swap-in must have been byte reversed.
/ • During the install process, byte reverse each doubleword of Little-Endian information on the media. / This format cannot be moved to an implementation with Bi-Endian memory.

A.1.3 Memory

The memory is composed of base memory and optional upgradable memory. The system board has socket(s) for JEIDA 88-pin IC DRAM cards or 72-pin SODIMMs (small outline dual inline memory modules) for upgrade memory.

Both base and optional memories are +5-volt, 32-bit and support data parity but do not support ECC. In power-down modes, the memory is refreshed at a slow rate to sustain data.

A.1.4 PCI Bridge/Memory Controller (PCIB/MC)

The portable system uses a custom-designed controller that bridges between the 603, memory, and PCI buses.

A.1.5 Power Management Controller

The portable system uses a custom-designed controller that performs low-level power management and PCI bus arbitration. System ROM is connected through this controller to the PCI bus.

A.1.6 Flash ROM -- 512 KB (AMD Am29F040-120)

This component contains the POST and boot code. It is recommended that the system board speed, native I/O complement, and like information be programmed into this device. There is no other source of such information built into the system board. Algorithms for programming this data into Flash before or during the manufacturing process are possible.

After power on, the processor fetches the initial code from this device. A maximum of 16 MB for System ROM and registers is allowed in the Reference Implementation memory map, but the system board utilizes a 512-KB device for the System ROM.

A.1.7 SCSI Controller -- NCR 53C810

This 8-bit SCSI-2 controller supports 32-bit word data burst transfers with variable burst lengths as a PCI master device. A high-level SCSI processor requires very little 603 overhead in operations. A 50-pin single-ended SCSI connector is available. On the SCSI bus, a hard disk drive pack and CD-ROM drive are connected. The bus supports data transfer speeds of up to 5 MB/s and 10 MB/s in asynchronous and synchronous modes, respectively.

A.1.8 PCI-ISA Bridge -- Intel 82378ZB System I/O

In addition to ISA bus bridging, this controller provides the following system services:

- Two 83C37 DMA controllers
- 82C54 timer
- Two 8259 interrupt controllers
- System control ports

The PCI-ISA bridge positively decodes PCI cycles for selected addresses, and subtractively decodes unclaimed PCI cycles for the internal ISA bus.

The BIOS Timer address range is relocated to avoid conflicts with the EEPROM, which resides in ISA address range 0078h through 007Ch. The PCI bus arbitration is achieved by the power management controller, but not by the PCI-ISA bridge, on the portable system.

A.1.9 Native I/O Controller -- National Semiconductor PC87322 Super I/O

This ISA device contains a floppy disk controller, serial port controllers, and a parallel port controller.

The floppy disk controller is downward compatible to µPD765 and N82077. Through a 26-pin port, an external floppy disk drive is supported.

On this portable system, one of two serial port controllers is used and connected to a 9-pin D-sub connector. The controllers are compatible to PC16550s.

The bi-directional parallel port controller is compatible to IEEE 1284, and is connected to a 25-pin D-sub connector.

A.1.10 Extended I/O Controller

The portable system uses an I/O controller, which supports variable I/O functions. Most of system control registers are contained in this controller.

A PS/2-compatible auxiliary device controller provides two channels of serial interfaces: one for alphanumeric devices, the other for pointing devices. The serial interface for alphanumeric devices is directly and solely connected to the universal micro control unit (UMCU). The UMCU is equivalent to an 84-, 85-, or 89-key space-saving keyboard. The UMCU has an additional serial interface for external alphanumeric devices.

A serial interface is connected to pointing devices. A keyboard/mouse port provides serial interfaces for external alphanumeric devices and pointing devices.

The extended I/O controller supports a PS/2 Type-A EEPROM (64x16) for storing security passwords, vital product data, and some configuration information.

A.1.11 Real-Time Clock -- Dallas Semiconductor DS1585S

This ISA device has a Real-Time Clock in a 128-byte CMOS address space and an 8-KB NVRAM space.

A.1.12 Business Audio -- Crystal Semiconductor CS4231

Business Audio is provided through the Crystal Semiconductor CS4231 stereo audio integrated circuit. This device is the same IC used in the Reference Implementation and is described in Section 6.3.4, “Audio Subsystem.” One microphone and a pair of stereo speakers are provided as built-in features. Also supported are four 3.5-mm jacks for:

- Stereo earphones
- Stereo microphone input
- Stereo line in
- Stereo line out

Also, the conventional (Timer 2) PC speaker functions, UMCU (Universal Micro Control Unit) beeps, PCMCIA audio, and CD audio functions are supported for audio application and alert.

A.1.13 PCMCIA Controller -- Ricoh RF5C366C

This ISA device supports PCMCIA 2.0 sockets. The sockets can provide +5 volts and +12 volts as Vpp, and support +5-volt signal interfaces.

A.1.14 LCD Display

The LCD display has a minimum 640x480 resolution.

A Western Digital 90C24A2 graphics controller is connected to the internal PCI bus through a bus bridge and supports 1 MB of Video RAM. The graphics subsystem supports an 18-bit RAMDAC that provides for connection through a standard 15-pin D-sub connector of an external CRT display.

A.1.15 L2 Cache

A look-aside L2 cache is mounted on the PowerPC 603 bus.

A.2 Energy-Managed Workstation

This appendix gives a basic hardware overview of a PowerPC Reference Platform-compliant energy-managed workstation. Figure 44 shows a diagram of the components of this energy-managed workstation. The basic hardware consists of a system board, power supply, diskette drive, SCSI disk drive, CD-ROM, cables, keyboard, monitor, and mechanical package. This energy-managed workstation configuration will use the PowerPC 603 processor, and will provide a SCSI external port, two serial ports, one parallel port, business audio with a stereo microphone jack, stereo earphone jack, line in and line out jacks, video graphics, keyboard and mouse jacks, and four PCMCIA slots. There are no ISA or PCI expansion card sockets. Base memory support is 16 MB with expansion capability to a maximum of 80 MB of System Memory. L2 cache or processor upgrades are not described, but provisions for them are included in the configuration. Video is provided by a PCI-based S3-928 daughter card. The subsections below define the major components of this system.

A.2.1 System Board

The system board contains most of the electronics for the system. Major subsystems are connected to main memory through the PCI bus. The memory controller, System I/O, and SCSI are located on this bus. The video is attached to the PCI bus through a special socket, allowing for future upgrades.

The board is designed with a space-saving layout, using both a CPU card and a base I/O card. The boards require +5 volts and +3.3 volts to power most of the components. PCMCIA Flash programming features which require +12 volts are also supported by the power supply.

A.2.2 PowerPC 603 Processor

The PowerPC 603 processor is available in operating frequencies of 50, 66 and 75 MHz. There are 32 address bits and 64 or 32 data bits. It can be configured as either a 32-bit data device or a 64-bit data device via a pin. The 603 contains a 16-KB internal cache which is split between instruction and data. The 603 package uses 240 pins and requires +3.3-volt power.

The PowerPC 603 processor has a number of differences from the PowerPC 601 processor. Please consult the 603 documentation for details. Two changes in particular should be noted: the PowerPC 603 does not snoop the internal instruction cache, and for the PowerPC 603 and any PowerPC processor other than the PowerPC 601, the Endian switching instructions are different than those shown in the desktop Reference Implementation. Refer to Section A.1.2, “PowerPC 603 Processor,” for suggested Endian switching logic.

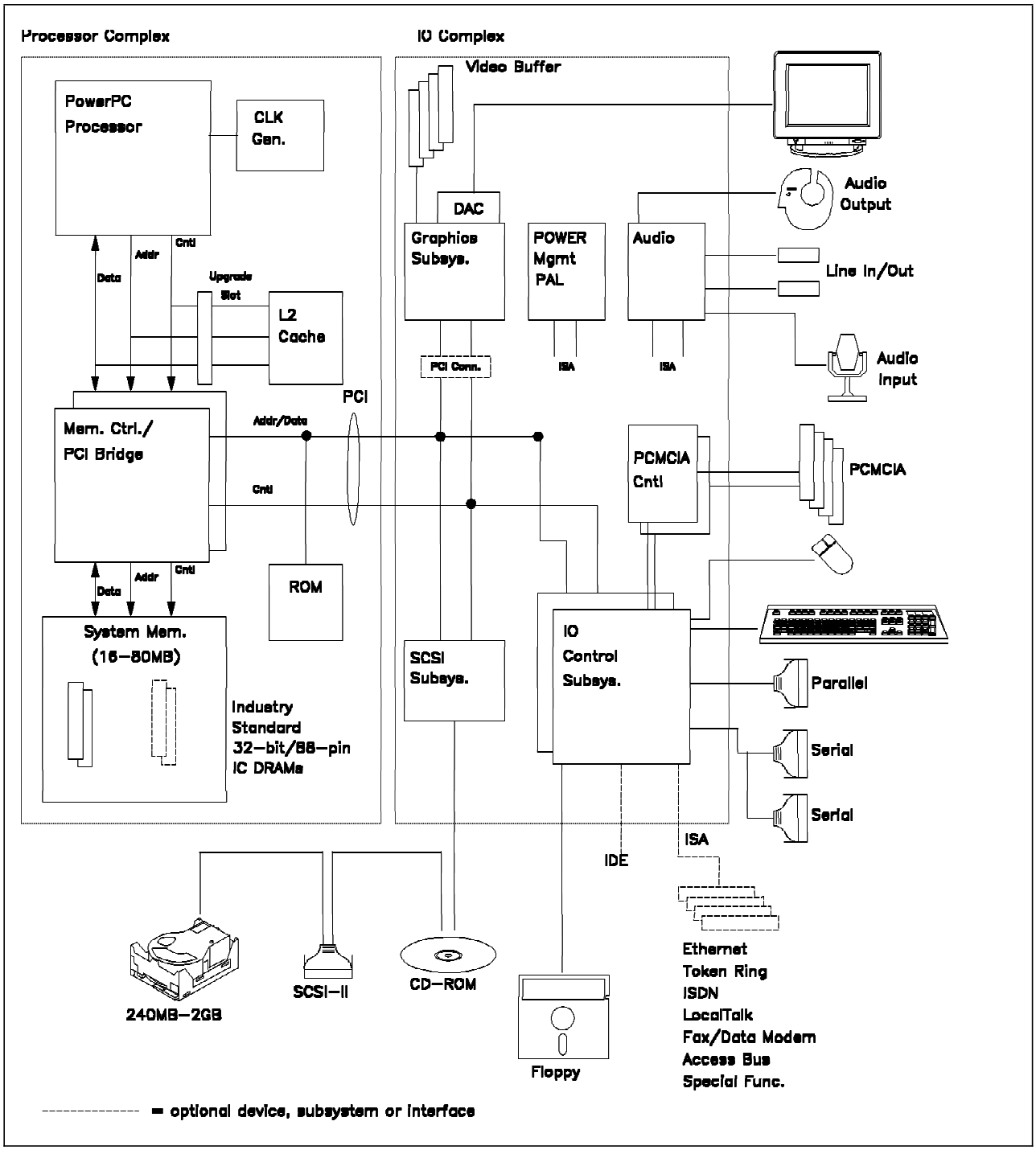


Figure 44. PowerPC Reference Platform Recommended Energy-Managed Workstation

A.2.3 Memory Subsystem

The memory subsystem is composed of base memory and optional upgradable memory. Base memory consists of 16 MB parity memory soldered directly on the system board. Upgrade memory consists of 2 JEIDA 88-pin IC DRAM card sockets. The memory controller has the following features:

- Does not support ECC memories
- Supports only 4-, 8-, 16- and 32-MB IC DRAM Cards

- Supports only non-interleaved memory access operation
- I/O Memory on the PCI bus is not cachable by the processor

A.2.4 PCI Bridge/Memory Controller (PCIB/MC)

The energy-managed system uses a custom-designed memory/bridge controller which provides the following functions:

- Processor bus cycles converted to the following:
  - System Memory cycles
  - PCI I/O cycles
  - PCI memory cycles
  - PCI interrupt acknowledgement
  - PCI configuration cycles
  - Address translation consistent with the memory map of the Reference Implementation
- PCI master access to System Memory
- Processor bus snoop cycles as a result of PCI System Memory access
- System Memory programmability for flexible memory device support

A.2.5 System I/O Bridge Chip -- SIO (Intel 82378ZB)

This function is provided by an Intel 82378ZB. It provides a PCI-to-ISA bus bridge where the native I/O and ISA attachments reside and it provides system services such as DMA and interrupt control. Its major functions are listed below.

- Bridge between PCI and ISA
  - Eight- or 16-bit ISA devices
  - Twenty-four-bit addressing on ISA
  - Partially decodes Native I/O addresses
  - Unclaimed PCI memory address below 16 MB -> ISA
  - Unclaimed PCI I/O address below 64 KB -> ISA
  - Powers up to an “open” condition
  - Generates ISA clock, programmable /3 or /4 divide ratio
  - Allows ISA mastering and has programmable decodes which map ISA memory cycles to the PCI bus
  - Has 32-bit posted memory write data buffer, no I/O buffering
- Seven-channel DMA between ISA devices and PCI memory
  - Eight- or 16-bit devices on ISA (and PCMCIA) bus only
  - Thirty-two-bit addressing at DMA
  - Function of two 83C37’s
  - Eight-byte bidirectional buffer for DMA data
  - Supports Scatter-Gather DMA operations
  - Supports Guaranteed Access Time Mode for DMA and ISA masters
- Timer block -- function of an 82C54
- Interrupt controller -- function of two 8259’s
- PCI bus arbiter -- unused in the energy-managed system
- Functions as PCI slave during programming and ISA slave cycles, as bus master during DMA (or ISA master cycles)

A.2.6 Native I/O Controller -- National PC87322 SUPER I/O

Control is provided via the ISA bus. This component contains:

- Floppy disk controller -- software compatible with DP8473, 765A, and NS82077.
- Two serial ports -- software compatible with INS8250N-B, PC16550A, and PC16450. The serial ports contain FIFOs.
- One parallel port -- bidirectional, EPP compatible.
- IDE interface -- unused on the energy-managed system.

A.2.7 SCSI NCR 53C810

This component attaches directly to the PCI bus on the system board and supports the following features:

- Eight-bit SCSI-2 interface
- Variable block size and scatter/gather data transfers
- Thirty-two-bit word data bursts with variable burst lengths
- Full 32-bit PCI bus master
- Sixty-four-byte FIFO buffer

A.2.8 Real-Time Clock (RTC) -- Dallas Semiconductor DS1585S

Functions of the RTC component include:

- RTC Function (PC compatible)
- Eight-KB Non-Volatile RAM
- Separate, replaceable battery
- Power management wake alarm interrupt

A.2.9 Keyboard/Mouse -- Intel 8042AH

This component contains the keyboard and mouse controls and is connected to the X bus (a buffered 8-bit subset of the ISA bus).

A.2.10 Flash ROM -- 512 KB (AMD Am29F040-120) / EPROM Alternative

This component contains the POST and boot code. It is recommended that the system board speed, native I/O complement, and like information be programmed into this device. There is no other source of such information built into the system board. Algorithms for programming this data into Flash before or during the manufacturing process are possible.

| After power on, the processor fetches the initial code from this device. A maximum of 16 MB for System
| ROM and Registers is allowed in the Reference Implementation memory map, but the system board utilizes
| a 512-KB device for the System ROM.

A jumper is provided which allows use of an EPROM (AMD 27C040-150JC -- 512 KB) in place of the Flash for systems that do not wish to bear the cost of Flash. The jumper must be present on all boards. Alternatively, 256-KB EPROMs may also be utilized.

A.2.11 Clock Generation

The 603 processor has a very simple clock requirement; therefore, the system board clock circuits are much simpler than in the Reference Implementation described in Section 6.2.5, “Clock Generation.”

The PowerPC 603 processor requires a single system clock input. This input sets the frequency of operations for the bus interface. Internally, the 603 uses a phase-lock loop (PLL) circuit to generate a master clock for all of the CPU circuits. This PLL circuit is locked to the clock input and may be set as an integer multiple of the clock input.

A.2.12 Business Audio

Business audio is provided through the Crystal Semiconductor CS4231 stereo audio integrated circuit. This is the same IC used in the Reference Implementation and is described in section 6.3.4, “Audio Subsystem.” Conventional (Timer 2) PC speaker functions are also provided, as well as support for audio from the PCMCIA connectors. Also supported are four rear-mounted 3.5 mm jacks for:

- Stereo earphones (the speaker is muted when an earphone jack is inserted into the connector)
- Stereo microphone input
- Stereo line in
- Stereo line out

There is also a system-board-mounted connector (with cable) for direct playback from the CD-ROM.

A.2.13 Timer 2 Audio Support

The Timer 2 signal is summed with the outputs of the CS4231 audio IC at the operational amplifier which drives the speaker. This provides the capability of supporting standard audio. Software may elect to directly drive Timer 2 audio or to emulate Timer 2 audio through the CS4231 audio IC.

A.2.14 Processor Time Base Support

The 603 processor’s Time Base function operates at one-fourth of the processor bus clock rate, which may be either 25 or 33 MHz. This Time Base and decrementer function is distinct from the calendar and RTC function normally found in PCs.

A.2.15 L2 Cache and Upgrade Processor Support

The system board has support for an upgrade socket, but no upgrade boards are defined for an L2 cache or processor upgrade for this system. The 603 bus signals and various control signals such as presence detect signals and cache control signals are wired to the socket. Support is included on the system board for:

- Write-through or copy-back cache cards. (Copy-back cache support is tentative at this writing.)
- Two cache type ID bits in a system board register.
- Various cache controls such as flush and disable.
- Arbitration for the upgrade processor or copy-back cache.

A.2.16 I/O Decoder

This component resides on the X bus. It receives partial decode signals from the SIO chip and further decodes these to produce chip selects for various components. It also contains most of the system registers and implements password protection.

A.2.17 Bi-Endian Support

The 603 processor normally operates with Big-Endian (BE) byte significance. It has a mode of operation designed to more efficiently process code written with Little-Endian (LE) byte significance. The system board has hardware to support Bi-Endian operation. The system defaults to BE mode at power on. The
/ mode may be switched, but the hardware is not optimized for frequent mode switching. Refer to A.1.2,
/ "PowerPC 603 Processor," for a description of the Endian switching approach.

When the system is in Big-Endian mode, data is stored in memory with BE ordering. When the system is in Little-Endian mode, data is stored in memory with LE ordering.

A.2.18 PCMCIA Controller -- Intel 82365SL

This configuration supports four PCMCIA option slots by using two Intel 82365SL PCMCIA controllers. Each 82365SL controls two PCMCIA slots. The system board has address/data buffers and power control as required in the PCMCIA specification. PCMCIA cards may be hot-plugged. The system board provides for two front and two rear PCMCIA slots. Each slot may contain either a type 1, 2, or 3 card.

This configuration supports propagation of the RING INDICATE signal from the controller to the system board for the "wake up" function. This system also propagates the AUDIO signal to support PCMCIA cards having audio output.

Note: The system board does not currently support +3.3-volt card operation.

A.2.19 Power Management Controller

A custom-designed power management controller is used in this configuration and performs the following functions:

- Dynamic control of the processor clock
- System clock control
- Peripheral power and mode control
- I/O activity monitor
- Programmable power management interrupt
- System status save and restore
- System power management status control

Power management for the energy-managed workstation can put the system into one of four states. These states are diagrammed in Figure 45 and are described as follows:

- OPERATIONAL
  - Default State
  - Maximum system performance
  - All system resources are turned on and available for immediate use
  - 100% power consumption rate
- STAND-BY
  - Active power management on local device controls
  - Potentially reduced system performance at a given instance
  - Devices are powered on demand
- SUSPEND
  - Maximum system power savings after a short period of inactivity while maintaining the capability to instantaneously return to operations
  - Most devices are either in a low-power mode or actually powered off
  - The processor may have the clock stopped

  - System state information is maintained along with System Memory contents
  - Most interrupts return the system to operational
- HIBERNATION
  - The system enters HIBERNATION after a longer period of inactivity or a HIBERNATION event has occurred
  - Most devices are powered off
  - System state information is stored on the system hardfile for later retrieval
  - Resuming operation will take longer than SUSPEND due to the restoration of the system state from the system hardfile
  - Maximum power savings while still being able to wake up by some external event
- OFF
  - The AC power is physically removed from the system

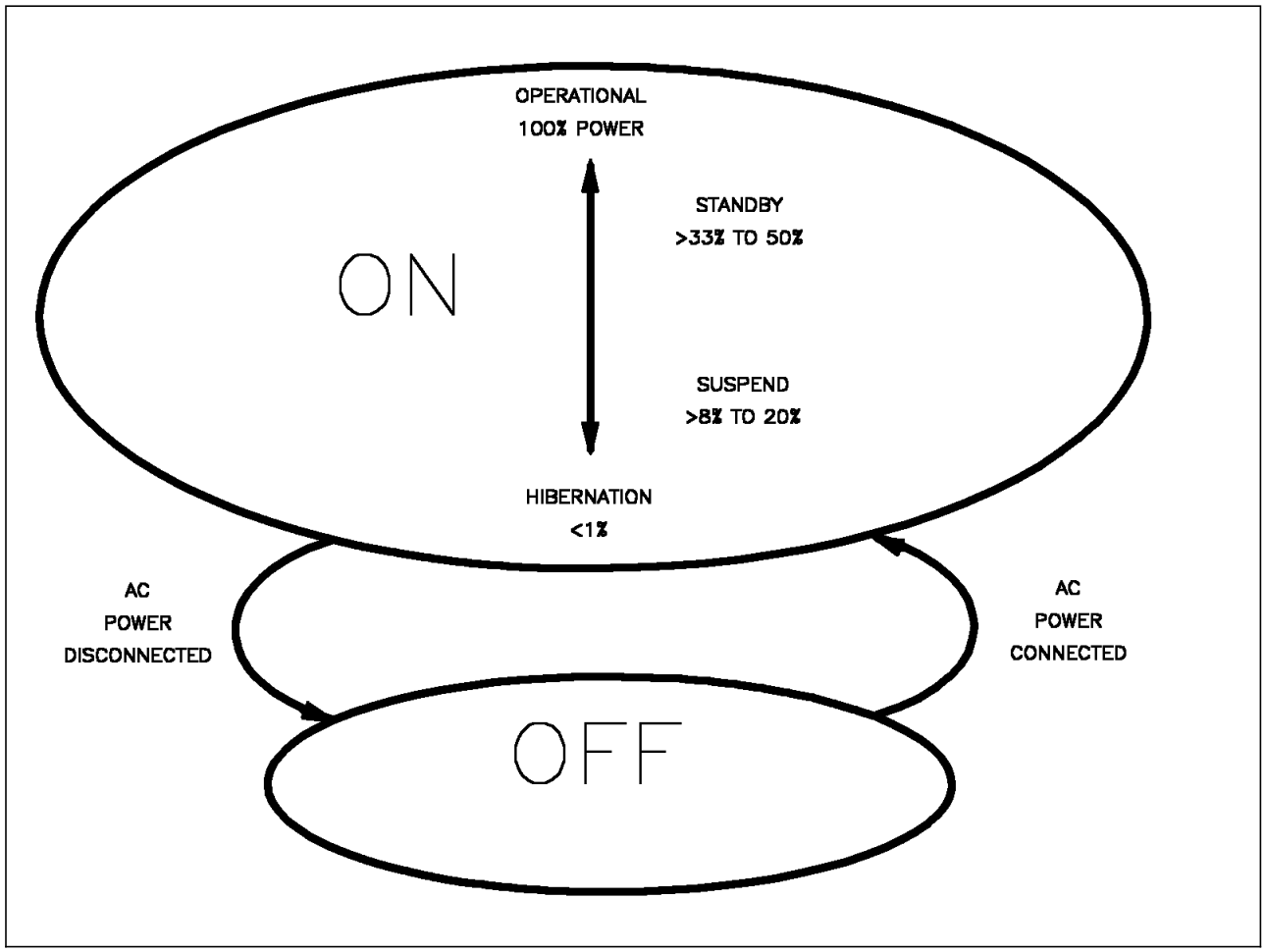


Figure 45. Energy-Managed Workstation's Power States

A.3 Medialess

The medialess configuration shown in Figure 46 is a simple variation on the Reference Implementation described in Section 6.0, "Reference Implementation." This configuration is different in two ways from the Reference Implementation:

a) The medialess system has no data storage capability. The SCSI subsystem, hardfile, CD-ROM, and floppy drive shown in Figure 18 would be removed from the Reference Implementation to form a medialess configuration.

b) The medialess system requires a network connection. The medialess system must be used attached to a network. The network would supply storage for the operating system including its boot-up code, for applications, and for data. An Ethernet or Token Ring adaptor would be placed on one of the expansion buses. The network would supply the operating system software, application software, permanent storage for the user's data and any temporary storage for paging.

The software which runs on the medialess system must accommodate this environment. The System ROM and the boot-up portion of the hosted operating system software must support boot from a network. The operating system must support the medialess operation and contain drivers for the network interface. Diagnostic software must be available through the network, or supplied by maintenance personnel (e.g. a floppy drive installed when needed for diagnostics).

A variation of the medialess system is a "dataless" system. In this system, a hardfile is placed in the configuration and used for system support (e.g. paging), but most of the software and data reside on the network. This configuration has performance advantages over a pure medialess configuration.

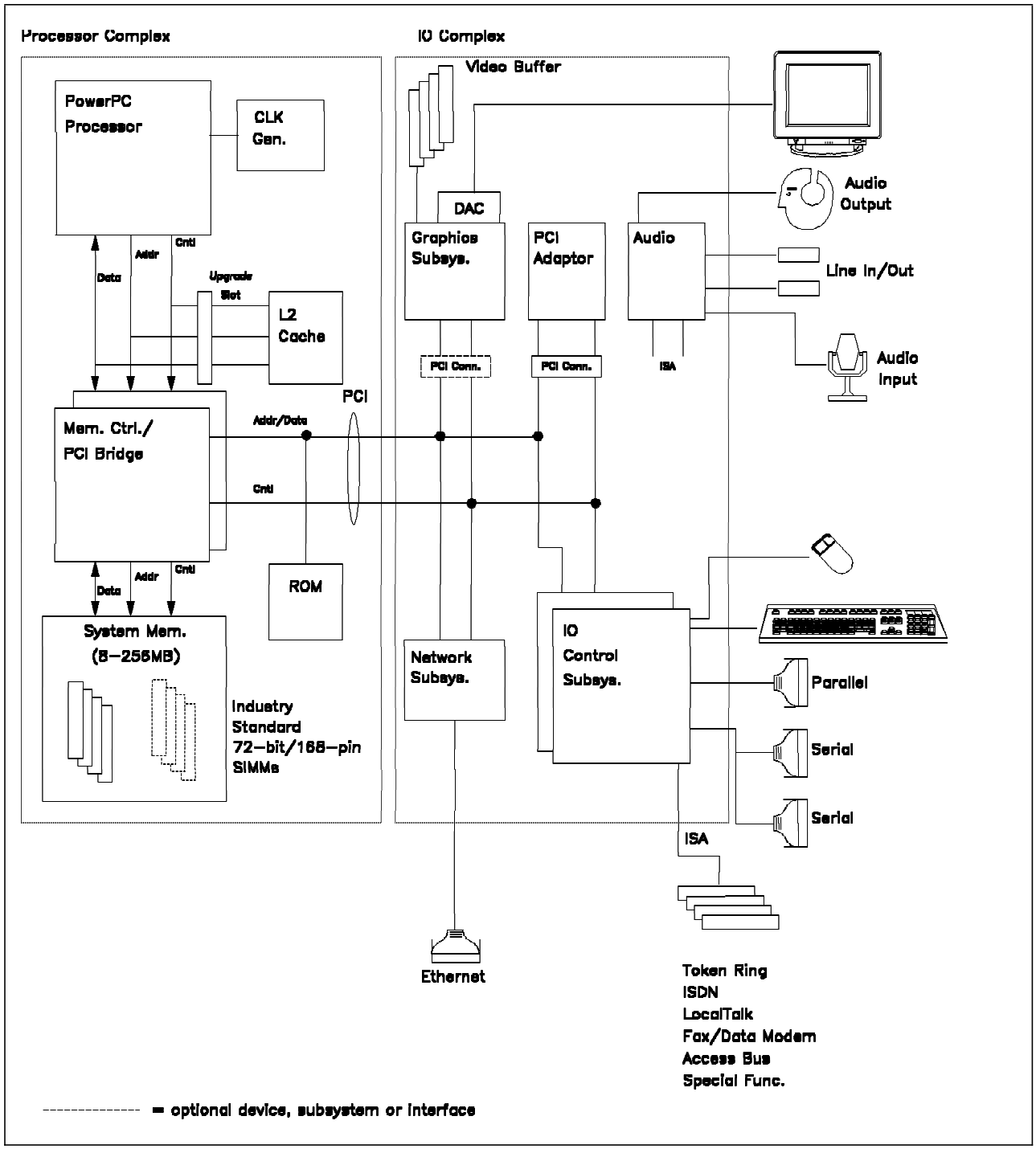


Figure 46. Recommended PowerPC Reference Platform Medialess System

A.4 Technical Workstation

The Technical Workstation configuration as shown in Figure 47 is a higher-performance variation of the Reference Implementation. It is intended to support graphics-intensive work, including 3D graphics. This configuration can differ from the Reference Implementation in several ways: increased system memory, an incorporated L2 cache, a second PCI bus, additional graphics resolution, or additional PCI adaptors.

The system memory for the technical workstation should be augmented. The memory should be upgradable to at least 128 MB. The memory system should implement ECC memory. The minimum system memory should be:

- 16 MB for an entry-level technical workstation
- 64 MB for an entry-level 3D graphics workstation

This memory system could be augmented by installing a copy-back L2 cache in the upgrade slot. For performance reasons, the graphics subsystem should be attached on a second PCI bus or attached directly on the local PowerPC processor bus. As shown in the figure, the graphics subsystem is attached to one PCI bus labeled “PCI 1.” A second PCI bus supporting the remainder of the I/O subsystems is labeled “PCI 2.” In this approach the memory controller and bus bridge would have to be changed from that used in the Reference Implementation (Section 6.2.3, “PCI Bridge and Memory Controller (PCIB/MC)”). If instead the graphics adaptor was placed on the processor bus, then the bus bridge and memory controller could be the same as the Reference Implementation, but the processor bus would have an additional load. The graphics system should support at least 1024x768 pixels with 8 bits of color information. Upgrades should be available for 1280x1024 pixels with 16 or 24 bits of color information. Two or three PCI adaptors are recommended. The number of adaptors depends upon the number of other loads connected to the PCI and the planned use of the workstation.

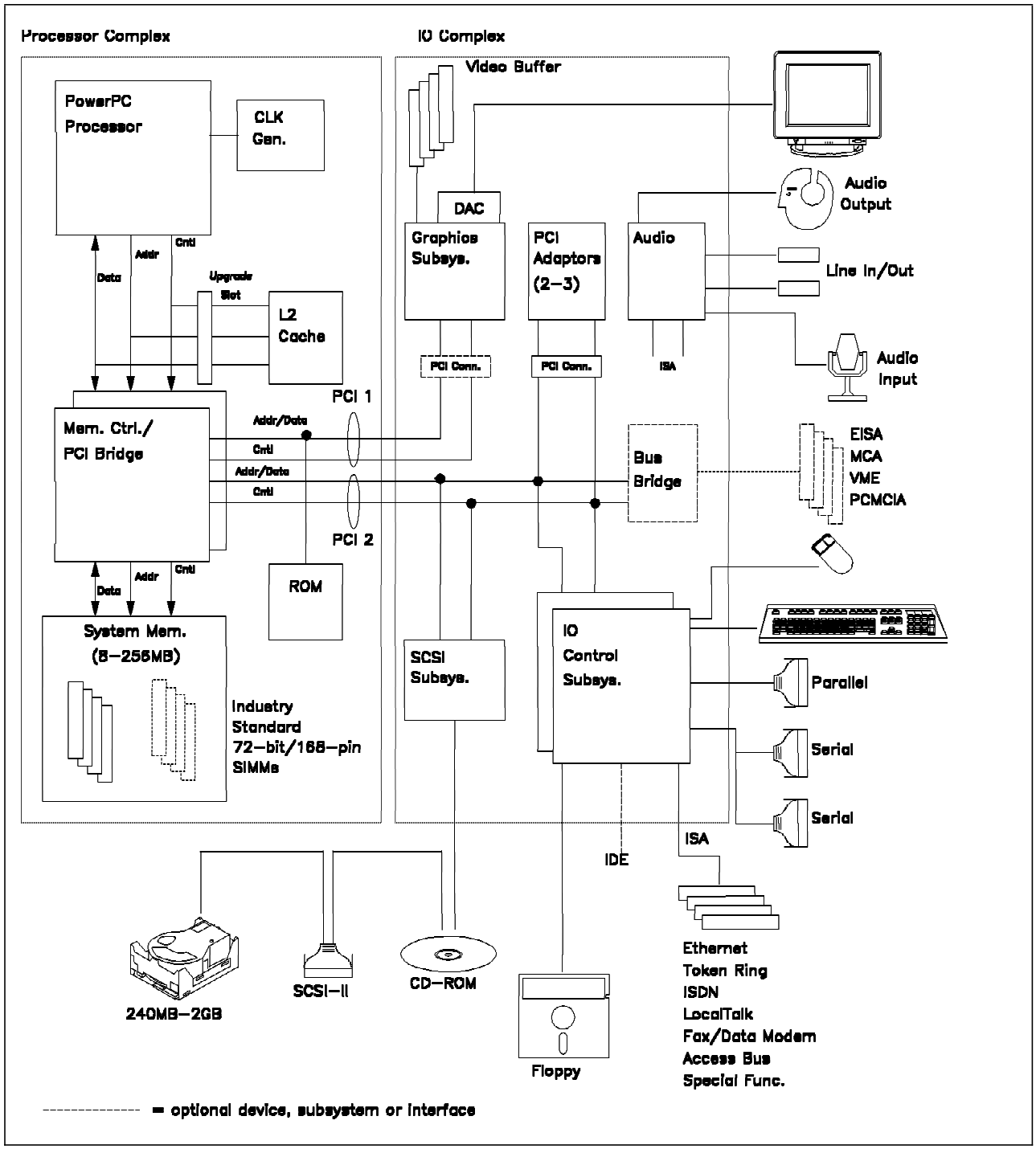


Figure 47. Recommended PowerPC Reference Platform Technical Workstation

A.5 Server

The server configuration section previously published in this specification has been deleted. A more comprehensive description of a server platform using a PowerPC processor will be included in a future version of this specification.

A.6 Symmetric Multiprocessor

The two-way multiprocessor configuration shown in Figure 48 is a variation of the Reference Implementation. The configuration differs from the Reference Implementation by having more processors, more L2 caches, more System Memory, multiple PCI buses, and additional facilities for interrupt handling and inter-processor communication.

/ Section 3.13, "Multiprocessor Considerations," lists the requirements and some of the recommendations for
/ interrupts, inter-processor communications, processor synchronization and memory coherence for multi-
/ processor systems. This section includes some additional recommendations and examples.

For multiprocessor environments, the *PowerPC Reference Platform Specification* intends to focus primarily on symmetric multiprocessor (SMP) systems. Figure 48 shows two processors connected through L2 caches to a system bus shared with the System Memory controller and PCI bus bridge. If the system bus is similar in design to the PowerPC processor bus, peak data bandwidths of greater than 400 MB/sec are possible. With an efficient L2 cache and memory system design, the system bus utilization for the two processors is manageable. SMP designs utilizing more than two processors may require greater system bus bandwidth and a more efficient memory system design (i.e. 16-byte data bus, GTL signal levels, 100-MHz bus clock, split address and data transactions, synchronous DRAM, interleaved memory, etc.).

In many system environments requiring multiple processors, additional performance is required in the I/O
/ subsystems. Often, the size and complexity of the I/O subsystem scales nearly linearly with the number of
/ processors. By using multiple bus bridges attached to System Memory and to the system bus, I/O operations can better utilize the available memory bandwidth. To optimize I/O-to-memory performance, bursting data transfers should be used, special data buffering within the bus bridge should be used, and key
/ I/O subsystems should have bus master capability so that they may manage data transfers independent of the system's processors.

Error detection, error isolation, error reporting and error recovery are important considerations in a multiprocessor system. Errors occurring in one processor or L2 cache should not be allowed to propagate to System Memory or to other processors or L2 caches. I/O adaptors and bus bridges should detect and report errors. ECC memory is recommended. Software should attempt to recover from some types of errors.

Multiprocessor configurations offer significantly greater availability than uniprocessors and are better suited for many applications than clusters of uniprocessors. The recommended recovery action, in the event of a processor or L2 cache failure, is to reset and reinitialize the failing processor. With appropriate software support, data retained in main memory and/or on disk can be used to roll back transactions, reestablish terminal sessions, re-dispatch tasks and re-try I/O operations associated with the failing CPU. It is important that the system continue to run even if some number of tasks are lost.

The following are some recommended attributes for a two-way SMP for a commercial database or file server environment:

- 1- to 4-MB L2 caches, in-line, copy-back, MESI protocol (per processor). One in-line L2 cache per processor is recommended because connecting two processors to one lookaside L2 cache would exceed the capacity of the processor bus in some, if not all, workloads.
- 64-MB minimum System Memory, expandable to 1 GB, ECC, synchronous DRAM support.
- An intelligent SCSI-2 subsystem which can efficiently handle a RAID-5 hard disk array.
- An intelligent communications adaptor with buffering (Ethernet, Token Ring, ATM, etc.).
- Multiple PCI buses attached through a single Memory Controller/ PCI Bridge subsystem.
- Enhanced Interrupt Controller with programmable redirection and priority capabilities. Must also handle inter-processor interrupts.
- All other system features recommended in the "Hardware Configuration" and "Reference Implementation" sections of this document.

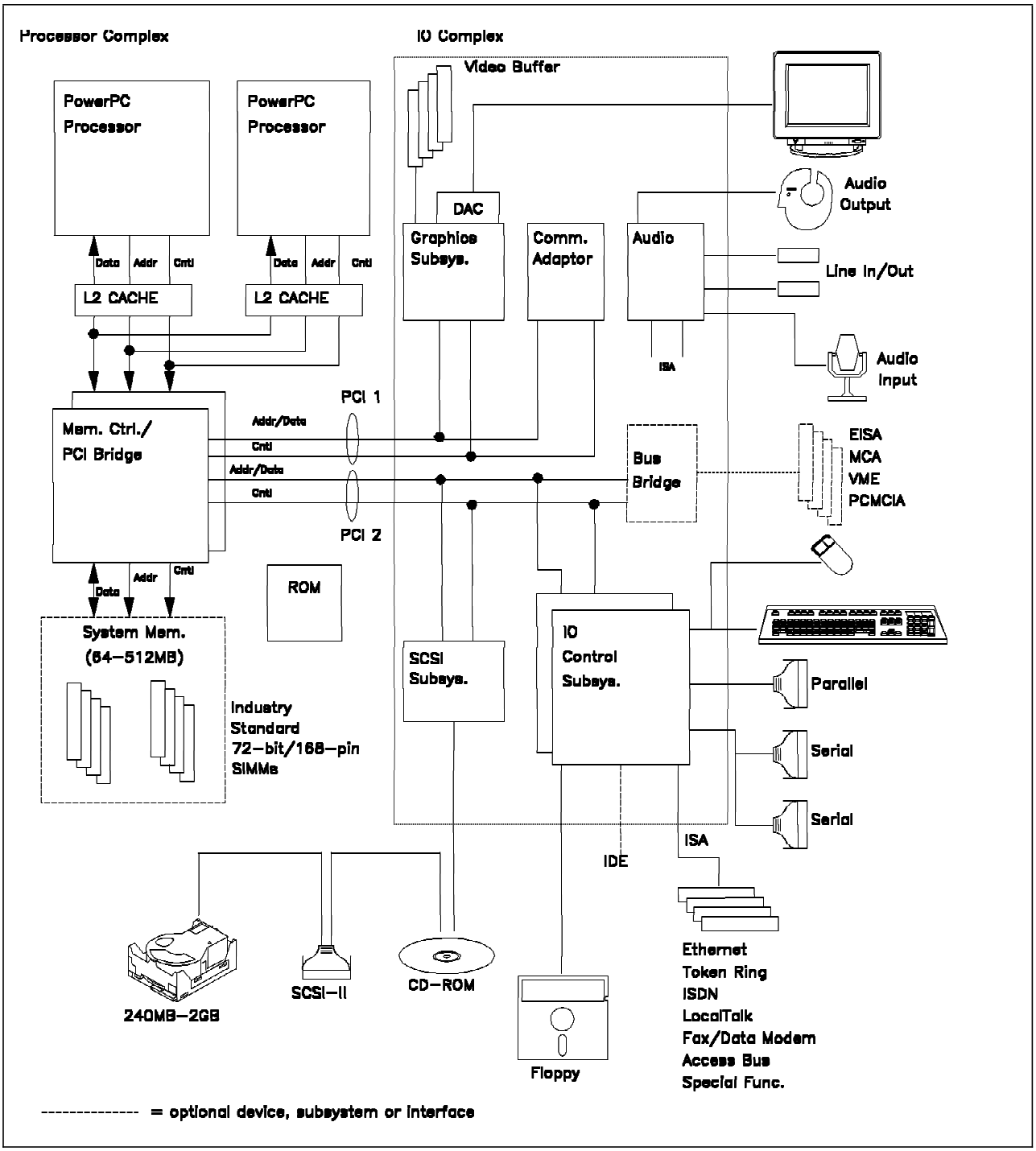
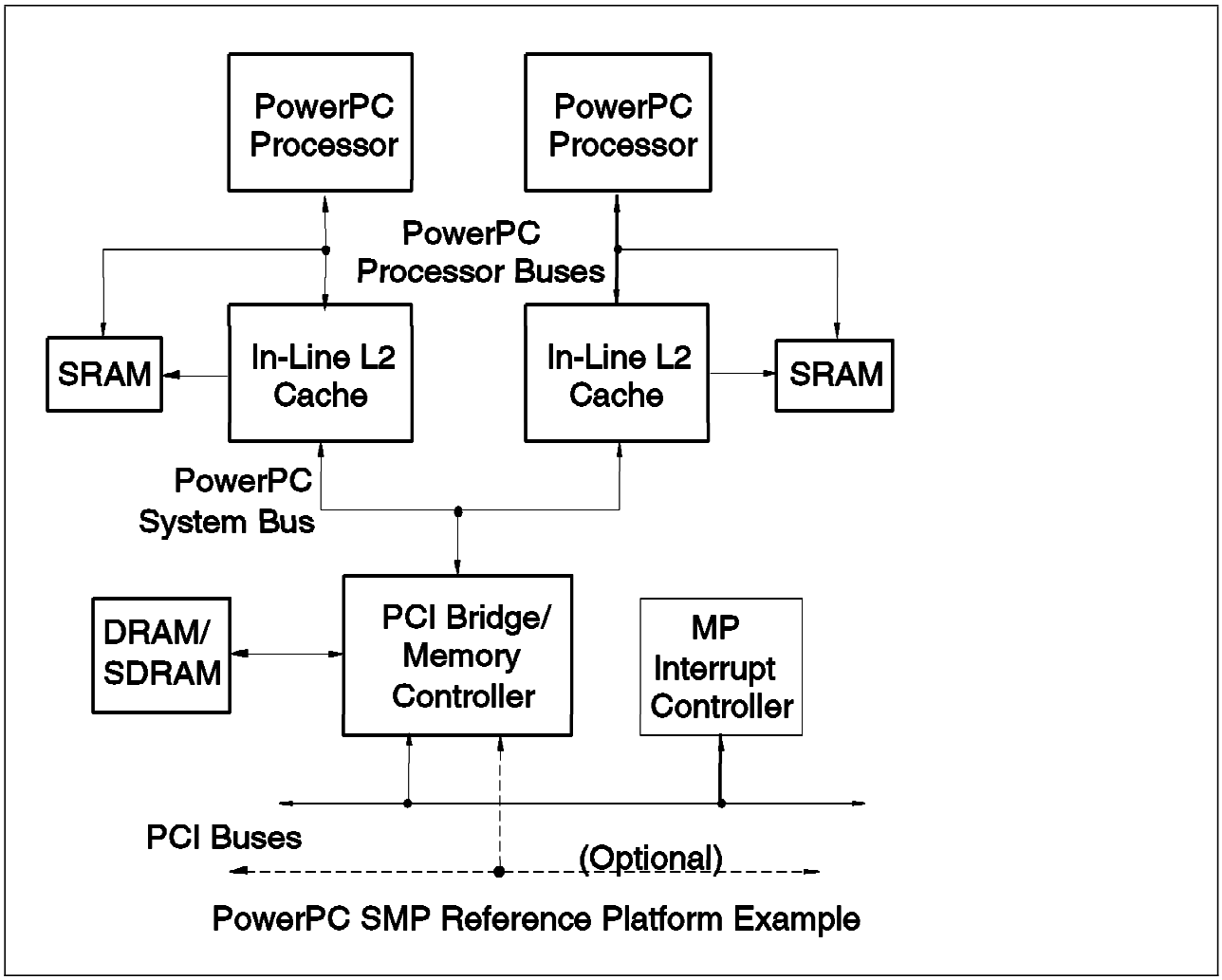


Figure 48. PowerPC Reference Platform Symmetric Multiprocessor System

/ A.6.1 SMP Overview

/ Figure 49 provides some additional detail on how an SMP system might be constructed. The in-line L2
/ cache, the memory controller/PCI host bridge, and the MP interrupt controller are each discussed in more
/ detail in the following sections. These sections briefly describe the basic components of each of these subsys-
/ tems along with general design guidelines.



/ Figure 49. SMP Overview -- Processor and Memory Subsystems

/ A.6.2 In-Line L2 Cache

/ Figure 50 is an example of an in-line, copy-back L2 cache for PowerPC based systems. (Please see Section / 6.7, “Upgrade Slot Definition” for a description of look-aside L2 caches.) Some of the features of an in-line / L2 cache include:

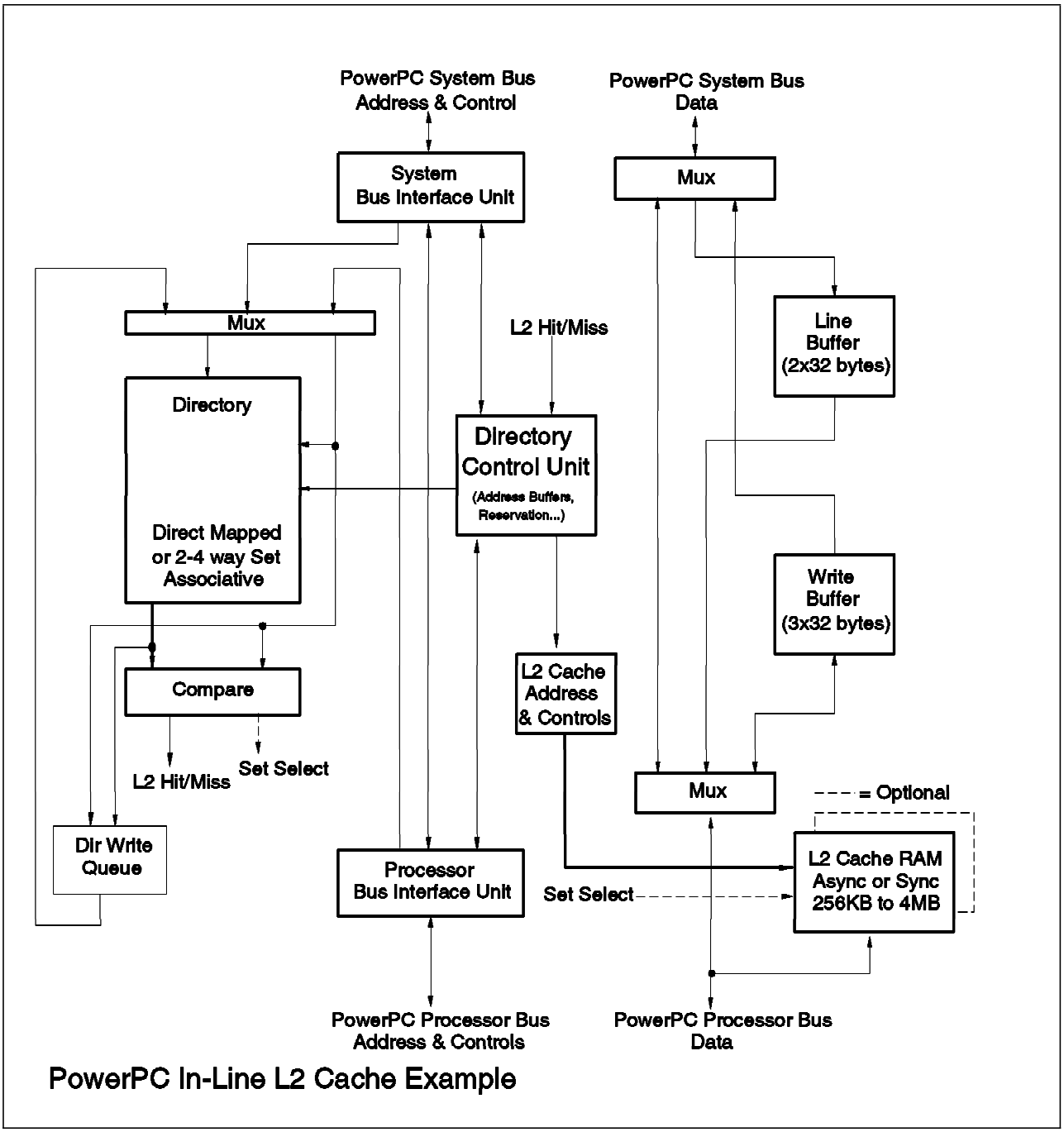
/ • Write Buffers: Three 32-byte write buffers are typical. (Each line in the 601’s cache contains two / 32-byte sectors. Each line in the 604’s caches contains one 32-byte sector.) If an L2 miss causes a line / to be cast out of the L2 cache, the old sector(s), if modified, are stored in the write buffers until they are / forwarded to the memory controller. Normally, the first sector of the old line being cast out from L2 / can be transferred to the memory controller before the first data from the new line arrives.

/ The write buffers are also used to hold write data from processor store requests which miss L2. These / processor store requests are typically tagged “write-through” or “cache-inhibited.”

/ The third write buffer is typically dedicated to hold the sector pushed out of L2 due to a snoop opera- / tion.

/ • Line Buffers: Two 32-byte line buffers are typical. The line buffers are used for memory read oper- / ations. If data arrives from memory and the processor bus is unavailable, the data is written into the line / buffers. If data arrives from memory and the processor bus is available, the line buffers are bypassed.

/ The write buffers and the line buffers include address and controls and participate in bus snooping. / Snoop requests which hit on these buffers (pipeline collisions) are normally retried.



/ Figure 50. In-Line L2

/ • Directory: The directory, or Tag RAM, keeps track of which lines/sectors are held in the L2 cache.
/ System bus requests which hit on the L2 cache are forwarded to the L1 cache, if necessary.
/ • Reservation Address Register: The reservation address register within the L2 cache tracks the reserva-
/ tion, if any, held by the processor. The reservation address register is set by the Read Atomic and *lwarx*
/ Reservation Set requests on the processor bus. The reservation is cleared when the reservation (RSRV_)
/ output signal from the processor is not asserted. System bus requests which hit on the reservation
/ address register are forwarded to the processor whether or not the line is valid in the L2 cache. Reserva-
/ tions are on a 32-byte boundary.
/ • Address pipelining: Address pipelining allows a new bus transaction to begin before the current trans-
/ action has finished by overlapping the data transfers associated with a previous request with the address

/ transfer for subsequent requests. In-line L2 caches typically support one level of address pipelining for each of the processor and system buses.
/ • Split transactions: Split transactions allow one bus master to have mastership of the address bus while another bus master has mastership of the data bus. In-line L2 caches typically support split transactions for both the processor and system buses.
/ • TLB Invalidate and *tlbsync* requests from the processor bus are forwarded to the system bus and vice-versa.
/ • *Sync* and *eieio* requests from the processor bus are forwarded to the system bus after the write buffers are flushed.
/ • The L2 cache is normally a superset of the L1 cache (except for certain special cases such as lines marked "write-through.") When a line is LRU'd out of L2, the corresponding line in L1, if any, is pushed out of L1 or invalidated, as needed.
/ • Snoop filtering: The L2 cache directory typically includes an inclusion bit to indicate whether a line currently resident in the L2 cache is also resident in the L1 cache. If a snoop request hits on the L2 directory and the inclusion bit in that directory entry is off (indicating the line is not in the L1 cache) there is no need to snoop the L1 cache. This speeds up snoop responses and cache-to-cache transfers.
/ • L2 cache line size: 32- or 64-byte line sizes are typical.

/ Typical configurations for L2 caches include:

/ • A direct-mapped 512-KB or larger L2 with an integrated cache control/directory and discrete synchronous SRAM chips.
/ • A 2-way set-associative 256-KB or larger integrated cache-control/directory/RAM single-chip cache.

/ L2 cache performance observations:

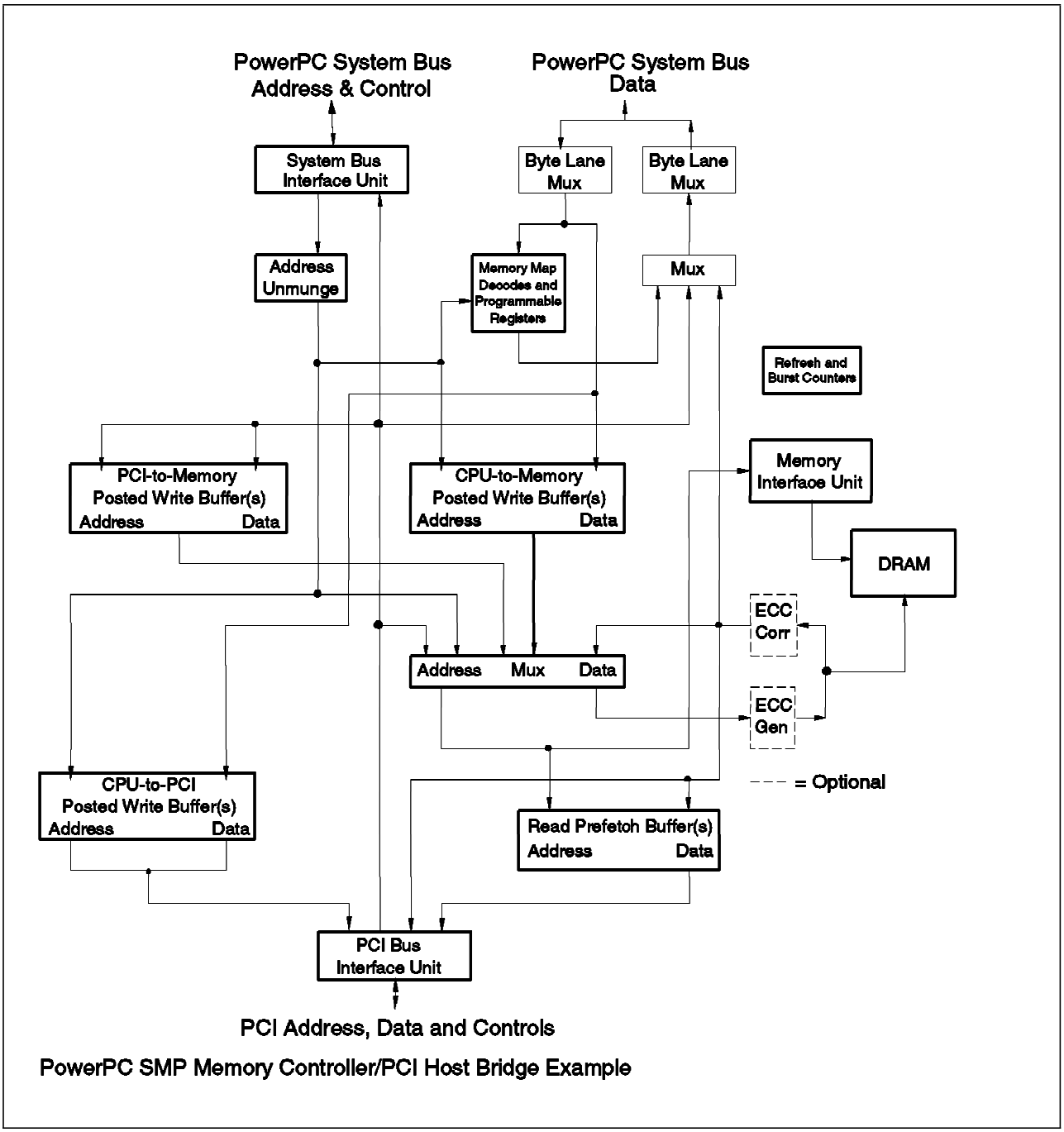
/ • Benchmark YOUR cache and YOUR memory subsystem design! A well-designed L2 cache can improve processor performance by 25-35%.
/ • Run the processor bus and the L2 cache at the highest possible frequency (66 - 80 MHz is typical).
/ • L2 cache read access times are normally stated in terms of number of cycles from transfer start (TS#) on the PowerPC processor bus to the first data beat (DH, DL) and number of cycles between data beats. For systems using synchronous SRAMs, 2-1-1-1 and 3-1-1-1 are typical L2 cache read access times (at 66 MHz). For systems using asynchronous SRAMs, 2-2-2-2 and 3-2-2-2 are typical L2 cache read access times. Avoid design trade-offs which add additional cycles (wait states) to the L2 cache read access times.
/ • For small L2 caches (256 KB), a multi-way set-associative L2 cache of size x performs almost as well as a direct-mapped L2 cache of size 2x.
/ • Use write buffers and line buffers to pipeline requests and free up the processor bus as quickly as possible.
/ • Use split transactions to overlap address-only requests (e.g. snoop requests) with data transfer.

/ A.6.3 Memory Controller/PCI Host Bridge

/ Figure 51 is an example of an integrated memory controller/PCI host bridge for PowerPC based systems.
/ Features include:

/ • CPU-to-memory posted write buffer. One 32-byte or one 64-byte CPU-to-memory posted write buffer is typical (should match L2 line size).
/ • PCI-to-memory posted write buffer: one or two 32-byte PCI-to-memory posted write buffers per PCI bus are typical.

/ The posted write buffers allow write requests on the system or PCI bus to be queued while main memory is busy servicing a previous write or read request. The posted write buffers also help permit simultaneous and independent operation of the system and PCI buses.



/ Figure 51. Memory Controller/PCI Host Bridge

/ Strict ordering must be preserved between the posted write buffers. All snoop activity must be resolved
/ before data from either the processor or PCI is written into the posted write buffers. For example, if a
/ PCI write request to System Memory hits on a line which is modified in a processor's cache, the line
/ from the processor must be pushed out of the processor's cache and placed into the CPU-to-memory
/ posted write buffer before the PCI request is placed into the PCI-to-memory posted write buffer (the
/ PCI request is typically re-tried until the data is pushed out of the processor's cache to the
/ CPU-to-memory posted write buffer). These requests should be written to System Memory in the order
/ they were placed in the posted write buffers.

/ There is no indication on the PowerPC processor bus whether a cache block write request is the result
/ of a snoop operation or some internal condition (e.g. LRU) within the processor or L2 cache. If the
/ memory controller/PCI host bridge (or the L2 cache) needs to determine that a cache block write

request is in response to a snoop request, the memory controller/PCI host bridge (or the L2 cache) must compare the address for the snoop request with the address for the cache write block request.

- The CPU-to-PCI posted write buffers free up the processor and the system bus for other activity while the PCI write operation proceeds independently. Typical configurations include:
  - Four 8-byte CPU-to-PCI write buffers.
  - Two 16-byte CPU-to-PCI write buffers.
  - One 32-byte CPU-to-PCI write buffer.

Snoop requests which hit on any of the CPU or PCI posted write buffers are normally retried.

- The read prefetch buffer, typically one 32-byte buffer, is used to hold data read from main memory for the last PCI read request. Subsequent PCI read requests to the same 32-byte sector access the read prefetch buffer, freeing up main memory for other requests. The read prefetch buffer is invalidated if a CPU or PCI write request hits on the buffer.
- The System Bus Interface Unit typically includes arbitration logic for three system bus masters (two in-line L2 caches plus the memory controller itself). Some systems, such as a four-way SMP, include arbitration for up to five system bus masters.
- The PCI Bus Interface Unit typically includes arbitration logic for six PCI bus masters (including the PCI host bridge itself) per PCI bus.

PCI (bus master) requests to and from System Memory must be maintained coherent. These requests typically cause a snoop operation on the system bus. PCI read requests are typically retried if the snoop request hits on a modified line in any processor's cache. PCI write requests are typically retried if the snoop request hits on a line in any processor's cache.

Sync and *eieio* requests from the system bus cause the memory controller/PCI host bridge to flush all posted write buffers.

The PowerPC system bus typically runs synchronous with the PCI bus at one, one and one-half, two, or three times (programmable) the PCI bus frequency. Some systems run the PowerPC system bus asynchronous with respect to the PCI bus; these systems support a wider variety of internal processor, processor bus and system bus frequencies at some increase in cost and complexity.

Performance observations for the memory controller/PCI host bridge:

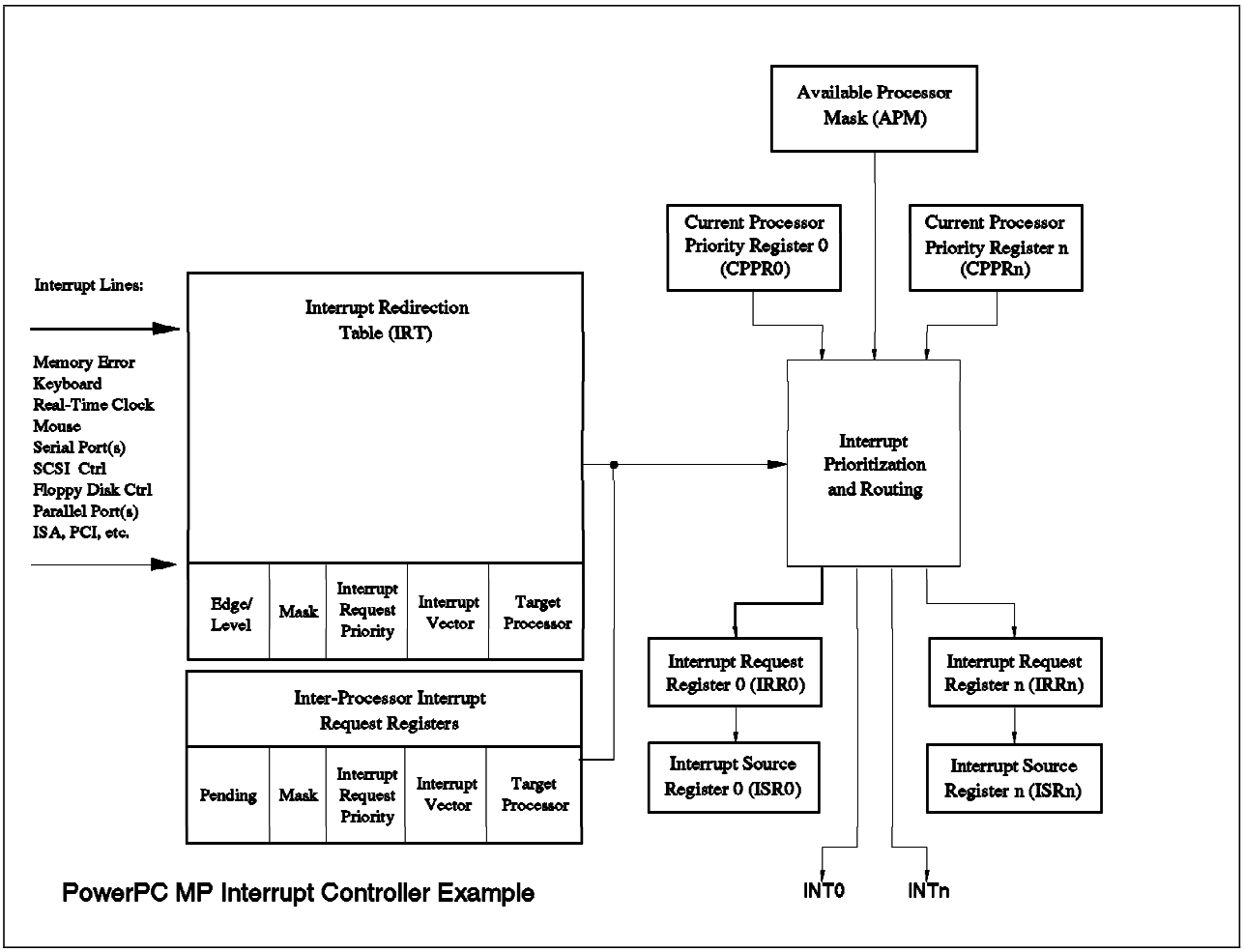
- Main memory read access times are normally stated in terms of number of cycles from transfer start (TS#) on the PowerPC system bus to the first data beat (DH, DL) and number of cycles between data beats. For systems using 70 ns synchronous DRAM (preferred), 7-1-1-1 is a typical main memory read access time (at 66 MHz). For systems using 70 ns asynchronous DRAM, 7-3-3-3 is a typical main memory read access time. (Write times, excluding the effect of the posted write buffers, are similar.)

A.6.4 MP Interrupt Controller Example A

Figure 52 is an example of a feature-rich MP interrupt controller. (Many other implementations are possible.) Features include:

One entry in the Interrupt Redirection Table (IRT) for each I/O interrupt. Each entry in the IRT includes:

- Edge or Level: 1 bit. (Edge-sensitive interrupts are converted, internally, to level-sensitive interrupts.)
  - 0 = edge-sensitive
  - 1 = level-sensitive
- Mask: 1 bit.
  - 0 enables the interrupt
  - 1 masks the interrupt



/ Figure 52. MP Interrupt Controller

/ • Interrupt Request Priority (IRQP): Typically 5 to 8 bits (32 to 256 interrupt priority levels). This is the
/ priority assigned, by software, to this interrupt. This field is optional; interrupt request priority can be
/ implemented in hardware or interrupt priority can be resolved through a software routine.
/ • Interrupt Vector (IRV): Typically 8 bits (256 interrupt vectors). This is the interrupt vector assigned,
/ by software, to this interrupt.
/ • Target Processor: Typically 2 to 8 bits. The interrupt is normally routed to the specified target
/ processor. A target processor of all ones (e.g. 'FF'X) enables priority-based interrupt routing (see
/ below).

/ The Inter-Processor Interrupt Request Registers (IPIRRs) are used for inter-processor interrupts. The field
/ definitions in the IPIRR are the same as for the Interrupt Redirection Table except that the Pending bit
/ replaces the Edge/Level bit.

/ The IPIRR can be used with the Target Processor field set (by software) to all ones to generate a floating
/ system-wide interrupt request. The IPIRRs allow software to maintain system-wide interrupt queue(s) (one
/ or more per system) in addition to processor-specific interrupt queues (one or more per processor).

/ The Target Processor fields in the IRT and in the IPIRR are used to redirect an interrupt request to a spe-
/ cific processor or to all processors.

/ The Current Processor Priority Registers (CPPR - one per processor) are used to assign a priority level to
/ each processor. These registers are required only if priority-based interrupt routing is implemented.

/ For interrupts targeted to a specific processor, an interrupt request with an IRQP higher than that
/ processor's current priority interrupts that processor; interrupt requests with an IRQP equal to or less than
/ that processor's current priority remain pending until that processor's priority is lowered.

/ For interrupts using priority-based interrupt routing, only one processor (normally the first processor whose
/ priority (CPPR) drops below the interrupt request's priority (IRQP)) is interrupted. In the case of a tie, one
/ processor is arbitrarily selected.

/ (Note: If priority-based interrupt routing is not implemented, an interrupt request is routed to the specified
/ target processor when the interrupt request is both pending and enabled. A single interrupt mask bit per
/ interrupt request per processor would support this simplified implementation.)

/ The Interrupt Request Registers (IRR -- one per processor) are used to identify which interrupt(s) are
/ pending for each processor. The IRR can indicate either all pending interrupts for that processor or just the
/ highest-priority pending interrupt for that processor. Software can poll the IRRs to see which interrupts are
/ pending for which processors.

/ The Interrupt Source Registers (ISR -- one per processor) are used to identify the last interrupt(s) routed to
/ that processor. The ISR can include the interrupt vector(s), the interrupt level(s) awaiting service, or other,
/ model-dependent information. If software polls for interrupts, the ISRs may not be required.

| Normally, the highest-priority pending interrupt in the IRR is transferred to the ISR when the software
| interrupt handler issues an interrupt acknowledge request. (Interrupt acknowledge is decoded by the PCI
| host bridge.) The corresponding interrupt vector is returned to the processor that issued the interrupt
| acknowledge request. If no interrupt(s) are pending, interrupt acknowledge returns a model-dependent spu-
| rious interrupt vector, typically all zeros or all ones.

/ The Available Processor Mask (APM) is used to ensure that only those processors currently in the config-
/ uration participate in interrupt routing.

/ Interrupt prioritization and routing can be implemented many ways. One way is for hardware to periodically
/ cycle among all pending interrupt requests. The highest-priority pending interrupt(s) for each processor are
/ stored in that processor's interrupt request register. Interrupts which can be routed to more than one
/ processor are stored in the IRR corresponding to the processor with the lowest CPPR. If the highest-
/ priority pending interrupt in a processor's IRR has a higher priority than that processor's CPPR, the inter-
/ rupt (INT) line to that processor is asserted.

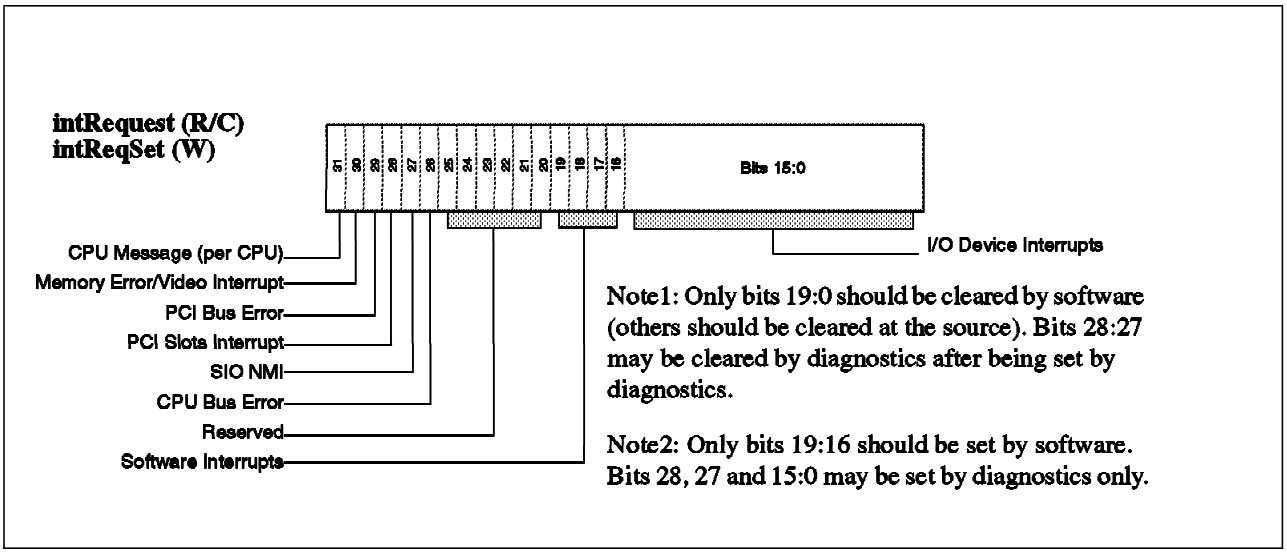
/ All the registers in Figure 52 can be read and written by software. All registers should be read and/or
/ written 32 bits at a time to minimize Bi-Endian design issues. Unimplemented bits return zeros on a read.
/ Reads, except for interrupt acknowledge requests, have no side effects.

/ A.6.5 MP Interrupt Controller Example B

/ This is an example of a lower-cost MP interrupt controller. This design meets all the requirements for MP
/ interrupts listed in Section 3.13, "Multiprocessor Considerations" but lacks some of the features of the pre-
/ vious example, most notably priority-based interrupt routing.

/ A.6.5.1 Interrupt Logic

/ The interrupt control logic must signal to the appropriate processor interrupts from I/O devices and the
/ memory subsystem, as well as software-initiated requests. An active interrupt source that is unmasked by a
/ processor will result in a pending interrupt for that processor. That pending interrupt asserts the appropriate
/ processor's interrupt line. That processor will then service the outstanding interrupt(s).



/ Figure 53. Global Registers (One per System)

/ Features of the implementation include the following:

/ • Capacity for up to 32 interrupt sources.
/ • Independent processor (per-processor) interrupt masks.
/ • Ability to let software prioritize interrupts by providing per-source masking as well as access to a register containing all unmasked interrupts.
/ • Per-processor message interrupts.
/ • Support for standard I/O device interrupts (software configurable).
/ • Support for four software interrupts.

/ A.6.5.2 Interrupt Handler

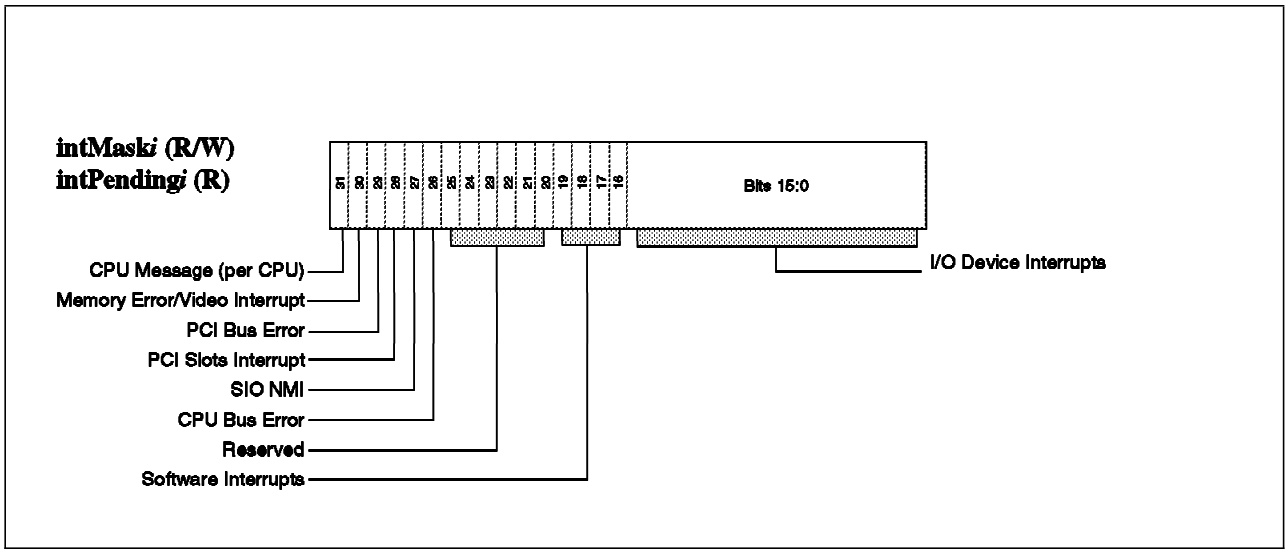
/ The software model for the interrupt handler relies on a set of software-accessible system registers. These registers are intRequest, intReqSet, intMaski, and intPendingi.

/ The implementation of the interrupt logic consists of a set of registers which define the source of each interrupt, a combinational logic per-processor interrupt generator and logic to handle reads and writes of the interrupt registers.

/ The system supports up to 32 interrupt sources. Figure 53 and Figure 54 show all possible sources for interrupts, as well as bit locations of each source. Each source has an associated bit in the intRequest register. Each of these interrupt sources has a level of masking associated with it at a per-processor level (called intMaski, where “i” is the cpu number, or cpuID). Interrupts which are unmasked propagate to the Interrupt Pending register associated with each processor (called intPendingi, where “i” is the cpuID). Whenever an Interrupt Pending register has a bit set, an interrupt is signaled to that processor. An example of this architecture is given in Figure 55.

/ Interrupt mask management in the per-processor Interrupt Mask Register (intMaski) is the responsibility of software. A suggested use is to mask specific interrupt sources to a specific processor (i.e. Memory Errors are only handled by processor 2).

/ There are program-settable message interrupt sources which are directed only at a particular processor. The interrupt hardware will direct the per-processor interrupt sources to the appropriate processors. These can then be masked off by the per-processor intMaski masks. Per-processor interrupts are not visible in the intRequest register since this register is global (not per-processor).



/ Figure 54. Per-Processor Registers (One per Processor)

/ A.6.5.3 Interrupt Request Set/Clear

/ The interrupt hardware allows software to set as well as clear the software interrupts in the intRequest register. The set function is provided by writing a “1” to the appropriate bit(s) of the intReqSet register. The clear function is provided by writing a “1” to the appropriate bit(s) of the intRequest register.

/ The I/O device interrupts correspond to the SIO IRQ0-15 interrupts. For every SIO Interrupt an IntAck cycle is generated on the PCI bus. The IntAck cycle will yield a 4-bit encoded packet from the SIO which corresponds to the highest-priority active SIO interrupt. This packet is decoded and the corresponding I/O device interrupt is set in the intRequest register. Additionally, diagnostics exclusively may set these interrupts by writing a “1” to the appropriate bit(s) of the intReqSet register. Once set, an I/O device interrupt will remain asserted until cleared by software writing a “1” to the appropriate bit(s) of the intRequest register.

/ Bits 31:29,26 always reflect the interrupt state of the source, and can only be set and cleared at that source. Bits 28:27 behave similar to bits 31:29,26, but can also be tested via diagnostics. Bits 28:27 may be set by writing the appropriate bits in intReqSet, but then MUST be cleared by writing to intRequest. Under normal conditions (non-diagnostics mode), these bits simply reflect the interrupt state of the source, and are set and cleared only at that source, not by software writes to intReqSet and intRequest.

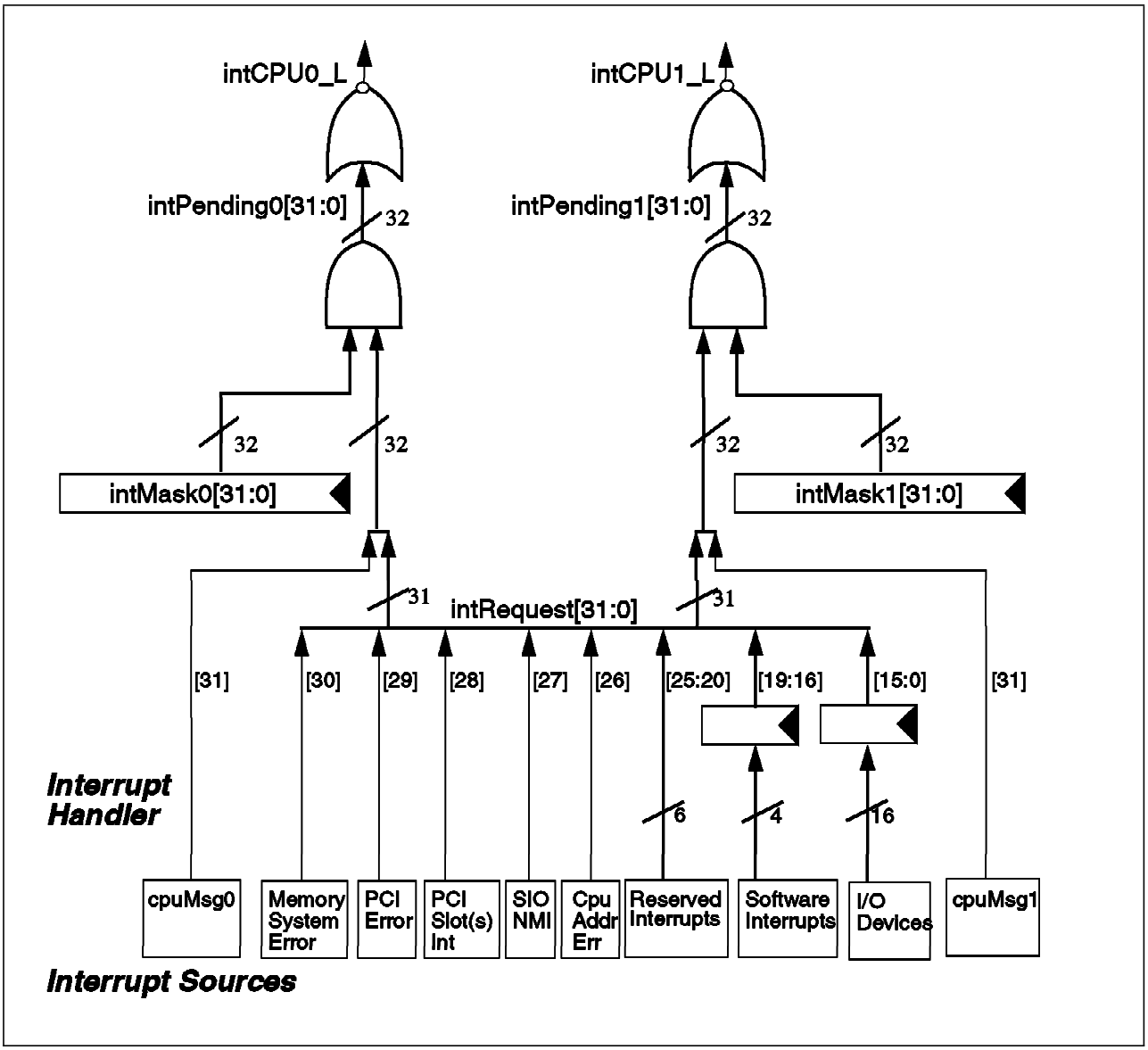
/ Bits 25:20 are reserved for future use, and should not be set or cleared.

/ A.6.5.4 Processor Interrupt Response

/ Once a processor receives an interrupt it is then software’s responsibility to read the appropriate Interrupt Pending register, determine the source of the interrupt, prioritize multiple interrupts, service the highest-priority interrupt at the source, and clear the corresponding interrupt request.

/ Unmasked interrupts which are reported through the intPendingi registers are updated every clock cycle. This means that after an interrupt has been signalled to a CPU, new interrupts may appear in the IntPendingi register. For example, it is possible for software to read the intPendingi register, mask out any asserted pending interrupts (via the intMaski register) and yet still have an asserted interrupt input, but from a different source requesting an interrupt since software last read intPendingi.

/ A sample interrupt handler process is as follows:



/ Figure 55. Interrupt Sources (Two-Processor Example)

/ a) Interrupted processor reads appropriate intPendingi register.
/ b) Processor branches to software interrupt handler which determines the highest-priority interrupt pending.
/ c) Source of highest-priority interrupt is pre-processed as necessary, then serviced as appropriate (e.g. device
/ driver code executed).
/ d) Device interrupt code returns to software interrupt handler for post-processing.
/ e) If source of request was bits 31:26, the request is cleared at the source; however, if source of request was
/ software or an I/O device, the bit is cleared in intRequest.
/ f) Software interrupt handler returns from interrupt.

A.7 A Proposed Diagnostic Strategy

This section describes a proposed diagnostic strategy for PowerPC Reference Platform systems. This proposal has not been adopted as a PowerPC Reference Platform architecture. It is included in this section as an example of an implementation approach which would be useful for high function systems (e.g. servers and multiprocessors). These systems would benefit from diagnoses of failing components and the resultant improved service time.

Terms used in this section include:

Error Status Registers Registers used by hardware to convey error information to firmware.
FRU Field Replaceable Unit
Fault Isolation The ability to isolate an error to a minimum number of FRUs.

Overview: When a hardware error occurs, firmware isolates the error to at most two FRUs and logs the error in NVRAM. The format of the error log in NVRAM is operating system independent. The operating system typically either reboots or stops the system after logging the error. The stand-alone diagnostics and, optionally, the operating system can subsequently be used to display the contents of the error log.

Operating systems may log additional operating system-specific error logs to NVRAM and/or to disk. These operating system-specific error logs are outside the scope of this section.

Recommendations

- It is recommended that systems implement sufficient hardware error detection and fault isolation logic to enable firmware to isolate hardware errors to at most two FRUs. Error status registers are used to convey this information to firmware.
- Time-of-failure fault isolation: When a hardware error causes a machine check interrupt, it is recommended that firmware analyze the error status registers and other information (e.g. device configuration tree) to determine:
  - which device is most likely source of the error and
  - which device reported the error.
- For PCI-related errors, it is recommended that firmware walk the device tree and read the following bits:
  - Data Parity Detected
  - Signaled System Error
  - Detected Parity Error

  in the device configuration header for the PCI host bridge(s) and the PCI devices to determine which PCI device is the most likely source of the error and which PCI device detected the error.
- While the operating system is normally involved in the processing of a machine check interrupt, it is strongly recommended that firmware not depend on an operating system to supply any of the implementation-dependent algorithms required for fault isolation.
- First and last error capture: It is recommended that firmware log each error in NVRAM. NVRAM contains a minimum of two error log entries.
- If all error log entries have been used, it is recommended that subsequent error logs overlay the most recent error log entry in a burst.

  Note: A hardware error which causes the processor to enter the checkstop state may or may not be logged in NVRAM.

- It is recommended that the system provide a way for firmware to reset the error status registers.
- It is recommended that the system provide a way for stand-alone diagnostics and operating systems to clear the error log entries in NVRAM.
- Standardized error log entry formats: It is recommended that a standard, operating system-independent error log entry format be defined for each class of errors (system board, memory, PCI, etc.).
- It is recommended that hardware systems which implement ECC memory record the failing memory address and syndrome when a correctable error occurs. It is recommended that correctable errors not

/ cause a machine check. Operating systems may periodically check for reports of correctable errors and
/ take appropriate action.

/ ***Miscellaneous***

/ • Examples of Field Replaceable Units (based on the Reference Implementation described in Section 6.0,
/ "Reference Implementation") include:
/ – System board
/ – Memory SIMM w
/ – PCI card in PCI card slot x
/ – ISA card in ISA card slot y
/ – Internal hard disk z
/ – L2 cache (and/or upgrade processor)
/ – Riser card
/ – Battery
/ – CD-ROM
/ – Flash ROM

Appendix B. Bi-Endian Design Guidance

This appendix gives design examples for Bi-Endian system implementations. These design guidelines are based on the current set of PowerPC processors (e.g. PowerPC 601, PowerPC 603, PowerPC 604). These processors assume that storage is Big-Endian. Therefore, in Little-Endian mode, a translation of data must be made somewhere in the system before the data reaches the PowerPC processor. The last sections discuss the design of a Bi-Endian graphics adaptor and the effect on these system designs if the processor architecture changes.

B.1 Little-Endian Address and Data Translation

In order for the current PowerPC processors running in Little-Endian mode to correctly access an object in Little-Endian-organized storage, the object must have its bytes show up in the correct processor byte lanes, and the Big-Endian address must be changed to refer to the correct location (after the object has had its byte lanes reordered). This translation has three parts.

The first part of the translation achieves address conversion and is done by the PowerPC processor. It is summarized below (a more detailed discussion can be found in an appendix of *The PowerPC Architecture*).

A PowerPC processor has status two bits that handle so-called Big-Endian and Little-Endian mode shifts. The Endian mode of the kernel and the Endianess of the current operating mode are recorded in two status bits (the register and bit definitions are implementation dependent and are found in PowerPC processor-specific user's manuals). Note that the PowerPC 601 uses only 1 bit. When Little-Endian mode is enabled, the effective address (EA) is modified in the PowerPC processor as shown in Table 24 before it is used to reference memory.

Table 24. Address Modification for Little-Endian Mode

Data Access	**EA Modifier**
Byte	XOR with b111
Halfword	XOR with b110
Word	XOR with b100
Doubleword	(no change)

This address modification results in correct Little-Endian addresses being presented to memory for aligned accesses (as will become clearer in the example below). The PowerPC architecture allows that unaligned accesses in Little-Endian mode trap, so incorrect memory references cannot be generated. (One subtlety here is that string operations and load and store multiple are considered unaligned accesses, and thus trap in Little-Endian mode also.)

The second part of the translation ensures that the bytes of the object addressed show up on the correct byte lanes of the processor. In Little-Endian mode, this translation requires that the bytes of a doubleword be reversed between I/O storage and the processor. This byte reversal may either happen as the data is read from or written to a Little-Endian I/O device and put into System Memory or it may occur as the data in Little-Endian order in System Memory is brought into the processor. The required byte alignment as a function of Endian mode and access is summarized in Table 25. The table assumes that the value X'1112131415161718' is stored at address 0 (actually any doubleword-aligned address.)

Table 25. Bytes Accessed Versus Endian Mode

	Byte Address								
Big-Endian	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Little-Endian**
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Byte at addr 0	11								Byte at addr 7
Byte at addr 1		12							Byte at addr 6
Byte at addr 2			13						Byte at addr 5
Byte at addr 3				14					Byte at addr 4
Byte at addr 4					15				Byte at addr 3
Byte at addr 5						16			Byte at addr 2
Byte at addr 6							17		Byte at addr 1
Byte at addr 7								18	Byte at addr 0
Halfword at 0	11	12							Halfword at 6
Halfword at 2			13	14					Halfword at 4
Halfword at 4					15	16			Halfword at 2
Halfword at 6							17	18	Halfword at 0
Word at addr 0	11	12	13	14					Word at addr 4
Word at addr 4					15	16	17	18	Word at addr 0
Doubleword at 0	11	12	13	14	15	16	17	18	Doubleword at 0
Instr at addr 0	i00	i01	i02	i03					Instr at addr 4
Instr at addr 4					i10	i11	i12	i13	Instr at addr 0

Thus a processor in Big-Endian mode that accesses the halfword at address 4 expects to see the value X'1516'. The most significant byte of the halfword, X'15', appears in byte lane 4 and the least significant byte, X'16', in byte lane 5. The table shows that applying the address mapping of Table 24 to the address and reversing the bytes between storage and the processor will result in referencing the quantity X'1516' in byte lanes 3 and 2 at halfword address 2 in a Little-Endian storage system, as required.

The third part of the translation requires that, in Little-Endian mode, addresses be generated correctly when addressing data in Little-Endian storage or when addressing I/O device addresses. For instance, since the effective address generated by the Little-Endian program as modified by the processor is used to access I/O, the programmer writing a device driver would have to precompensate the effective address used to access an adapter so that after the modification by the processor, the real address used to access the adapter is the real address of the target storage/register on the adapter. This translation must be done in every device driver and makes writing device drivers very error prone. A better solution is to solve the address modification problem one time in hardware. Logic must be added to the I/O interface such that when Little-Endian mode is selected, the added logic undoes the modification to the three low-order bits of the address. Then the unmodified (remodified) address used to access the I/O adapter is the same address as generated by the program. This system design is preferred, since a programmer writing a device driver is able to use the control register addresses as specified in the adapter hardware reference manual.

In summary, in a PowerPC Reference Platform-compliant machine, whenever the machine is running in an Endian mode different than the native mode of the processor (e.g. the PowerPC processor expects Big-Endian storage order, but the system is running in a Little-Endian mode) the address must be remapped and the byte lanes reversed somewhere on the way to or from the I/O subsystems.

B.2 Conforming Bi-Endian Designs

Several designs for implementing a Bi-Endian architecture are possible. The next two sections describe an approach in which both the memory and I/O subsystems are Bi-Endian, and a second approach in which memory is Big-Endian and the I/O subsystems are Bi-Endian.

B.2.1 Bi-Endian Memory and Bi-Endian Transportable I/O Design

The Bi-Endian Memory and Bi-Endian Transportable I/O design (e.g. the Reference Implementation) for a Bi-Endian system is shown in Figure 56. I/O devices are divided into two classes: "Transportable I/O," which consists of devices such as disks, tapes, and networks; and "Presentation I/O," which consists of devices such as graphics, audio, and video adaptors. For this design, memory, the processor and Transportable I/O interfaces must support both Endian modes. Data may exist on the Transportable devices in either Big-Endian or Little-Endian format; it is brought in, and sent out to the outside world, in either form. Presentation I/O are by design either Big-Endian or Little-Endian (or with extra hardware they may be Bi-Endian). The processor accesses both storage and I/O through a controlled byte reversal multiplexor and controlled address modification function (e.g. Xpose_bytes and Mod in Figure 56). The address modification algorithm is shown in Table 24. The byte reversal multiplexor reverses the position of each byte in a doubleword as shown in Table 26. Similarly, when the transfer between the processor and I/O is on a 4-byte I/O bus, the byte reversal is performed as shown in Table 27.

Table 26. Endian Mode Data Byte Reversal

I/O	**Storage Byte**	
Byte	**BE Mode**	**LE Mode**
0	0	7
1	1	6
2	2	5
3	3	4
4	4	3
5	5	2
6	6	1
7	7	0

Legend:

BE Mode Big-Endian Mode
LE Mode Little-Endian Mode

Table 27. Byte Reversal, Unequal Bus Widths				
I/O		**Storage Byte**		**Description**
Word	**Byte**	**BE Mode**	**LE Mode**	
0	0	0	7	Word 0: Even addressed word
	1	1	6	
	2	2	5	
	3	3	4	
1	0	4	3	Word 1: Odd addressed word
	1	5	2	
	2	6	1	
	3	7	0	

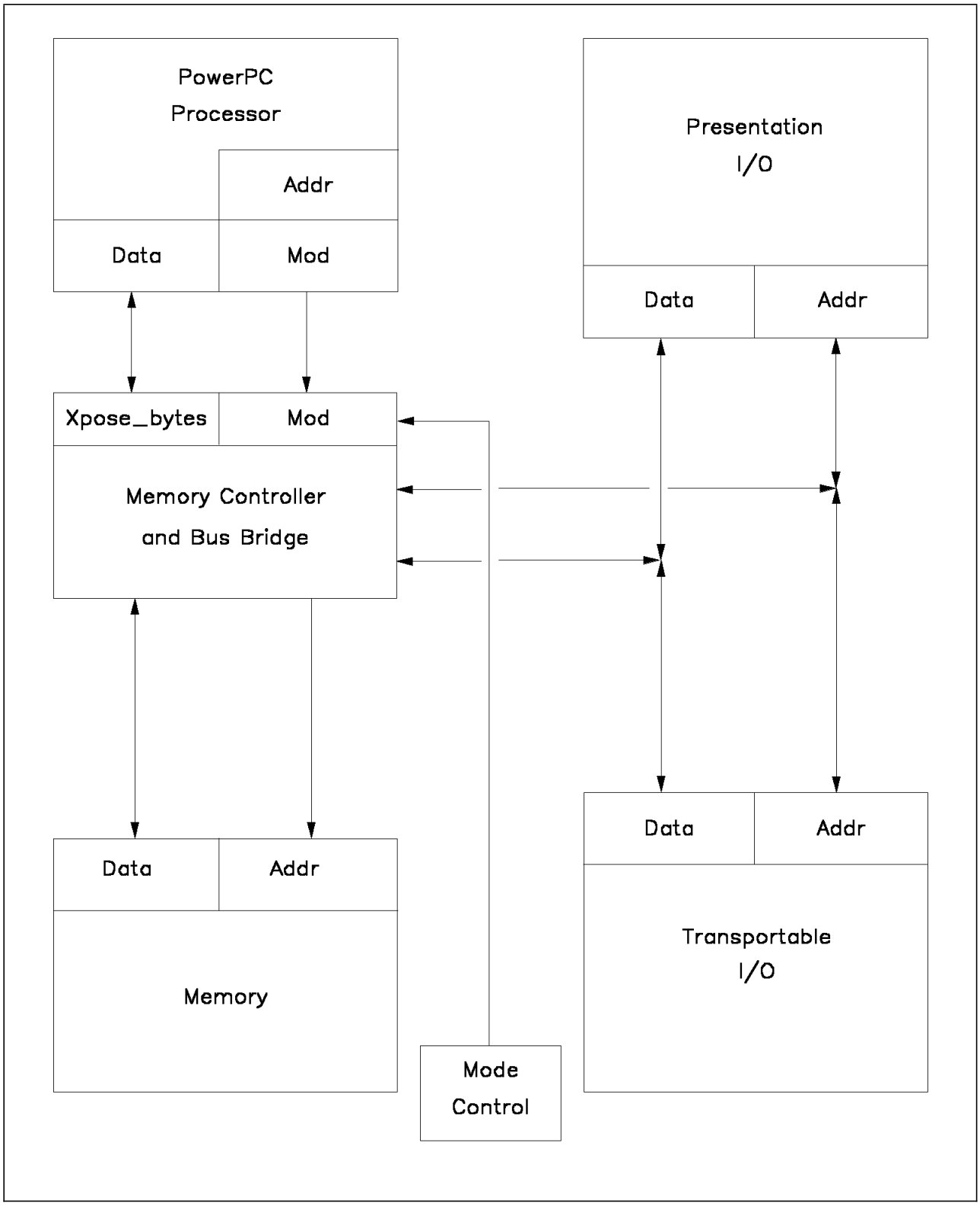


Figure 56. Bi-Endian System with Bi-Endian Memory and I/O

In this implementation, the memory and I/O are connected as if they were Big-Endian. In this case, byte 0 of storage or I/O is considered the most significant byte and passes through the controlled byte reversal multiplexor to byte 0 of the processor (MSB of the processor). The rules for applying the byte reversal multiplexor and the address modification are as follows:

Mode	Function
Big-Endian	No byte reversal of data and no address translation
Little-Endian	Byte reversal of data and address translation

From the viewpoint of Endianess of the data, I/O interfaces access only storage, not the processor or other I/O interfaces. I/O master transfers that move data from one device to another do not enter into the Endianess translation. With this in mind, areas that must be described to perform this Bi-Endian support include:

a) Processor and I/O mode control
b) Processor-to-memory interface
c) I/O-to-memory interface
d) Processor-to-I/O interface

The sections below describe these four areas. For the three interfaces, both address processing and data handling are described.

B.2.1.1 Processor and I/O Mode Control

The mode of the system may be changed by software, which has to perform the required functions to place the processor and I/O system in that mode. This design is based on the assumption that the processor and I/O are in the same Endian mode. No attempt has been made to architect a system where Little-Endian data is read and translated for a Big-Endian-mode processor running Big-Endian applications. The processor comes up in Big-Endian mode at power on or after a hardware reset. A definition in each PowerPC processor user's manual describes the process (sequence of instructions) required to change the Endian mode of that processor.

Since the processor does not provide an external signal indicating the Endian mode selected, the system must provide a mechanism to allow software to control the Endian mode of other subsystems. During configuration, software will use this mechanism to select the Endian mode to be used by storing the appropriate control value to the address of the control mechanism. The system design may place the control mechanism address in the real memory space or in the I/O Memory space, whichever is more convenient. Software must be able to address and alter the mode control mechanism in both Endian modes.

B.2.1.2 Processor-to-Memory Interface

In this form of the design, memory is assumed to be in the same Endian mode as the processor. Memory is accessed in two modes: as a result of cache-inhibited loads and stores or as a result of cache reads or writes. Each of these operations will be described below.

Operations between cache and memory are always doublewords or larger and as such the address in either Endian mode is unaffected. Data is byte reversed for Little-Endian mode and not byte reversed for Big-Endian mode. The result of these operations is that data brought into cache from Big-Endian memory is unchanged, and data brought into cache from memory in Little-Endian mode is byte reversed and stored as the PowerPC processor expects to see it (e.g. most significant byte first). Writing data back to memory from cache in Little-Endian mode reverses the byte order and restores the data in Little-Endian order.

Cache-inhibited loads and stores have to adjust the data and addresses for the expected processor formats of data and addresses. For Little-Endian mode, the address generated by an instruction points to where the data exists in Little-Endian storage. The address is translated by the processor, and then translated again outside the processor which returns it to its original value. The data is byte reversed, putting it in the order expected by the processor logic on fetches or expected in storage for stores. For Big-Endian mode the address is unchanged and the data is placed in memory in the order contained in the processor. The Refer-

ence Implementation approach would reverse the bytes once due to the connection to memory and then reverse them again in the multiplexor, restoring them to their original Big-Endian order.

B.2.1.3 I/O-to-Memory Interface

There are no address or data transformations between memory and I/O. Data is placed in memory in the same order as it exists on the I/O devices independent of Endian mode. The one exception might be Presentation I/O devices. Adaptors for these devices could be designed to accept data from Little-Endian and Big-Endian storage or to always accept data only in one format. In this case, the software would have to handle the data transformation. For example, a graphics adaptor that expected data in Little-Endian format would need software to byte reverse the data in Big-Endian mode before putting the data in memory.

B.2.1.4 Processor-to-I/O Interface

In Little-Endian mode, the address as modified by the processor must be modified again by the I/O interface such that the address used for the I/O access is the address computed by the storage instruction. Within the processor, the I/O addresses computed by a storage instruction are modified by the processor before the access is performed. Regardless of whether the access is to I/O space memory or a device control register, the address originally computed by the instruction is the address that must be used to access I/O space. The address modification algorithm shown in Table 24 is used to remodify this I/O address. This function is shown in Figure 56 in the box labeled “Mod,” which is controlled by the box labeled “Mode Control.”

The data transfer between the processor and I/O is managed in the same manner as the transfer between the processor and memory. Bytes are crossed between source and destination byte channels as indicated in Table 26 and Table 27. This byte transposition will occur once in Little-Endian mode to transform the processor-held data in Big-Endian format to Little-Endian format for I/O. In Big-Endian mode the byte transformation order is not transformed.

B.2.2 Bi-Endian I/O Design

The Bi-Endian I/O design for a Bi-Endian system is shown in Figure 57. For this design, storage is always Big-Endian, which means data is always stored with the MSB first. Transportable I/O interfaces must support both Endian modes. Data may exist on the Transportable devices in either Big-Endian or Little-Endian format; it is brought in, and sent out to the outside world, in either form. Presentation I/O are by design either Big-Endian or Little-Endian, and need not change modes since they deal with a constant visual and audio external world.

The processor accesses both storage and I/O. From the viewpoint of Endianess of the data, I/O interfaces access only storage, not the processor or other I/O interfaces. I/O master transfers that move data from one device to another do not enter into the Endianess translation. From a system viewpoint, I/O interfaces access only storage, not the processor or other I/O interfaces. Areas that must be designed to perform this Bi-Endian support include:

a) Processor and I/O mode control
b) Processor-to-memory interface
c) I/O-to-memory interface
d) Processor-to-I/O interface

The sections below describe these four areas. For the three interfaces, both address processing and data handling are described.

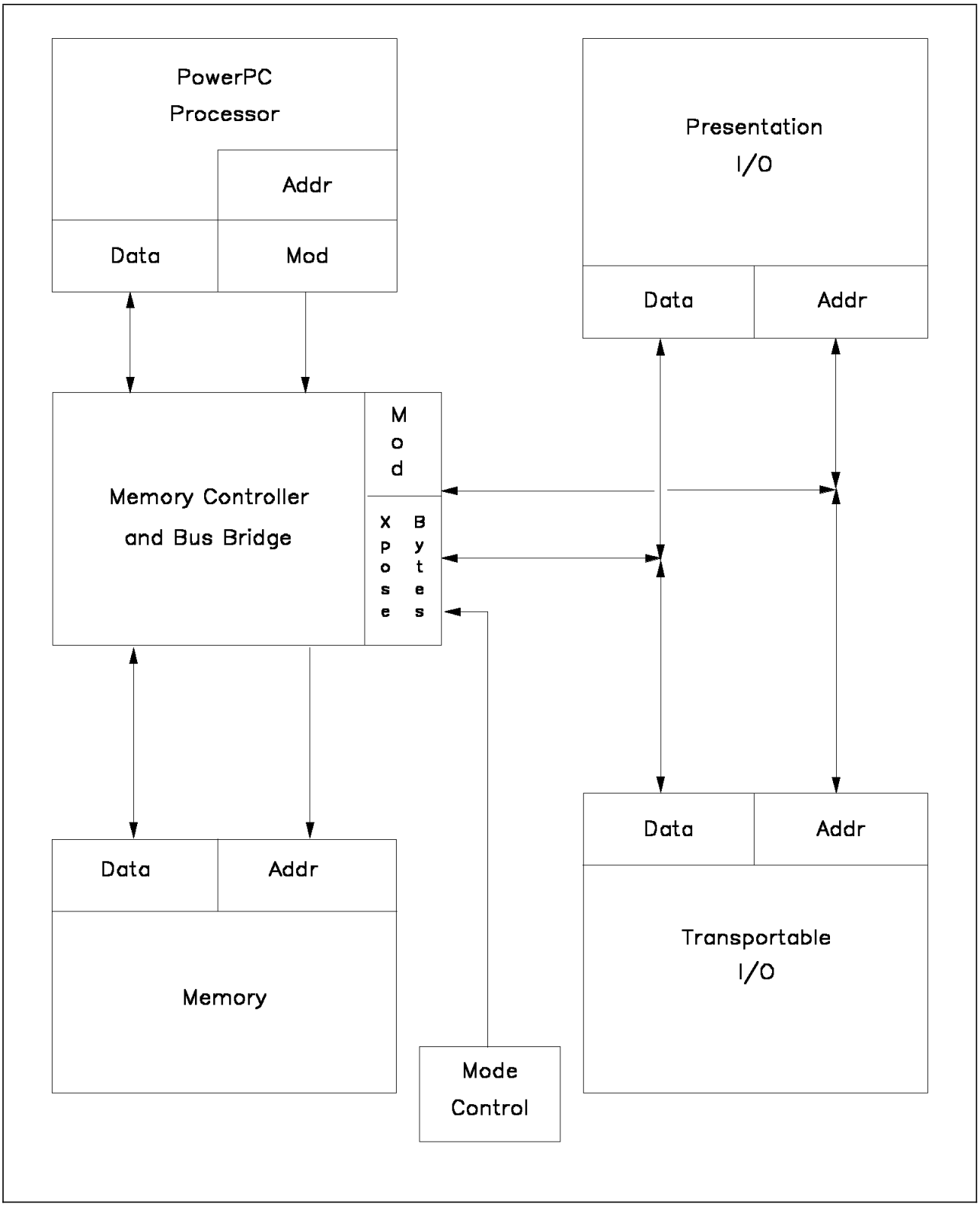


Figure 57. Bi-Endian System with Bi-Endian I/O

B.2.2.1 Processor and I/O Mode Control

The mode of the system may be changed by software, which has to perform the required functions to place the processor and I/O system in that mode. This design is based on the assumption that the processor and I/O are in the same Endian mode. No attempt has been made to architect a system where Little-Endian data is read and translated for a Big-Endian-mode processor running Big-Endian applications. The processor comes up in Big-Endian mode at power on or after a hardware reset. The definition in each PowerPC processor user's manual describes the process (sequence of instructions) required to change the Endian mode of that processor.

Since the processor does not provide an external signal indicating the Endian mode selected, the system must provide a mechanism to allow software to control the Endian mode of other subsystems. During configuration, software will use this mechanism to select the Endian mode to be used by storing the appropriate control value to the address of the control mechanism. The system design may place the control mechanism address in the real memory space or in the I/O Memory space, whichever is more convenient. Software must be able to address and alter the mode control mechanism in both Endian modes.

B.2.2.2 Processor-to-Memory Interface

Address translations are performed by the processor. Cache block transfers between the processor and storage are always a doubleword or larger and the address is not affected by the Endian mode. With Big-Endian storage, Little-Endian-mode loads and stores of aligned scalars with caching enabled work correctly after the address translation. When Little-Endian mode is enabled, loads and stores with caching inhibited use the address as modified by the processor. These instructions work the same and have the same constraints as for loads and stores out of cache.

For the processor-to- or from-memory interface, the data handling is independent of Endian mode. The memory stores multi-byte scalars in Big-Endian order, which has the most significant byte at the first byte of the address.

B.2.2.3 I/O-to-Memory Interface

In Little-Endian mode, storage addresses generated by I/O devices are modified using the address modification described in Table 24 prior to performing the access. This address modification is shown in the block labeled "Mod" on the right side of the Memory Controller and Bus Bridge. This address modification is performed for both Transportable I/O and Presentation I/O adaptors, as shown in Figure 57. This address modification is required to adjust for the format of the data in memory that has been byte reversed to place the MSB first.

Data transferred to and from the Transportable I/O devices that must switch Endian modes (e.g. disks, diskettes, communications) must have data that is stored in Little-Endian format converted to Big-Endian format. I/O transfers may be done at any implementation-determined width, but this byte reversal of data in Little-Endian mode is done by treating the data as a string of bytes that must be reversed within a doubleword (see Table 26). For unaligned transfers or transfers of less than a doubleword, bytes must be crossed from the source byte channel to the destination byte channel as shown in this table. This design places a byte-reversing multiplexor in the path from Transportable I/O to memory. This byte reversal multiplexor is shown as the "Xpose bytes" box in Figure 57.

When the interface between main storage and I/O requires a conversion from a two-word bus to a one-word bus, the byte reversal should be done in accordance with Table 27, which assumes a two-word bus to main storage and a one-word bus to I/O.

An unaligned access (e.g. read or write of System Memory) that crosses a doubleword boundary must be performed as multiple accesses. The address modification algorithm described above is not applicable to unaligned accesses. One approach to handling unaligned accesses is to perform the access as multiple aligned

accesses using byte, halfword, and word operations for which the main storage address is modified as described above.

The specific design of the Presentation I/O device adaptors will determine if a second byte-reversing multiplexor is present on the device and will influence how software device drivers must interface with this device. For instance, an audio adaptor that has byte reversal logic in it may be placed on a bus, while a graphics adaptor that does not have byte reversal logic may be on the same or a different bus. Depending upon the mode of the processor and memory, neither device may need byte reversal or both may need it. For example, a graphics adaptor with a Little-Endian design (e.g. registers and data are expected in Little-Endian order) could be addressed by a Big-Endian processor and by storage, via software performing the byte reversal, before the Big-Endian data was sent to the graphics adaptor. Alternatively, the graphics adaptor could be designed to perform the byte reversal. In this case, the adaptor would pass data straight through when the processor is in Little-Endian mode and do a byte reversal when the processor is in Big-Endian mode.

B.2.2.4 Processor-to-I/O Interface

In Little-Endian mode, the address as modified by the processor must be modified again by the I/O interface such that the address used for the I/O access is the address computed by the storage instruction. Within the processor, the I/O addresses computed by a storage instruction are modified by the processor before the access is performed. Regardless of whether the access is to I/O space memory or a device control register, the address originally computed by the instruction is the address that must be used to access I/O space. The address modification algorithm shown in Table 24 is used to remodify this I/O address. This function is shown in Figure 57 in the boxes labeled “Mod.”

The data transfer between the processor and I/O is managed in the same manner as the transfer between I/O and memory except that the processor is the master (it provides address and control) rather than an I/O mechanism. Bytes are crossed between source and destination byte channels as indicated in Table 26 and Table 27. This byte reversal is performed on all I/O between transportable devices and may be performed on I/O to presentation devices. The processor performs unaligned accesses as multiple accesses to aligned doublewords and may transfer an odd number of bytes within an aligned doubleword.

B.3 Software Support for Bi-Endian Operation

The Endian mode-switching logic should be a software abstraction provided by each operating system. The specific set of instructions are processor dependent.

As pointed out above, data for Presentation I/O and control data for any I/O device may have to be byte reversed. Services to perform these operations should be provided by the support software. Typically this would be language syntax and compiler support for a load and store with byte reversal of 2-, 4- and 8-byte scalars. If the compilers for all languages do not support these forms of loads and stores, then the operating system should supply services that perform the byte reversal.

B.4 Bi-Modal Devices

For performance reasons, applications in many operating environments write directly to graphics adaptors. It is recommended that graphics adaptors used in PowerPC Reference Platform systems provide both Big-Endian and Little-Endian data transfer methods. As of this document publication date, AIX requires a Big-Endian graphics adaptor interface, while all other operating systems that support the PowerPC Reference Platform require a Little-Endian graphics adaptor interface. It is recommended that graphics subsystems implementing bi-modal support use the design shown in Figure 58, which applies to both the frame buffer and the graphics subsystem’s register/command space. This is necessary in order to provide for adequate per-

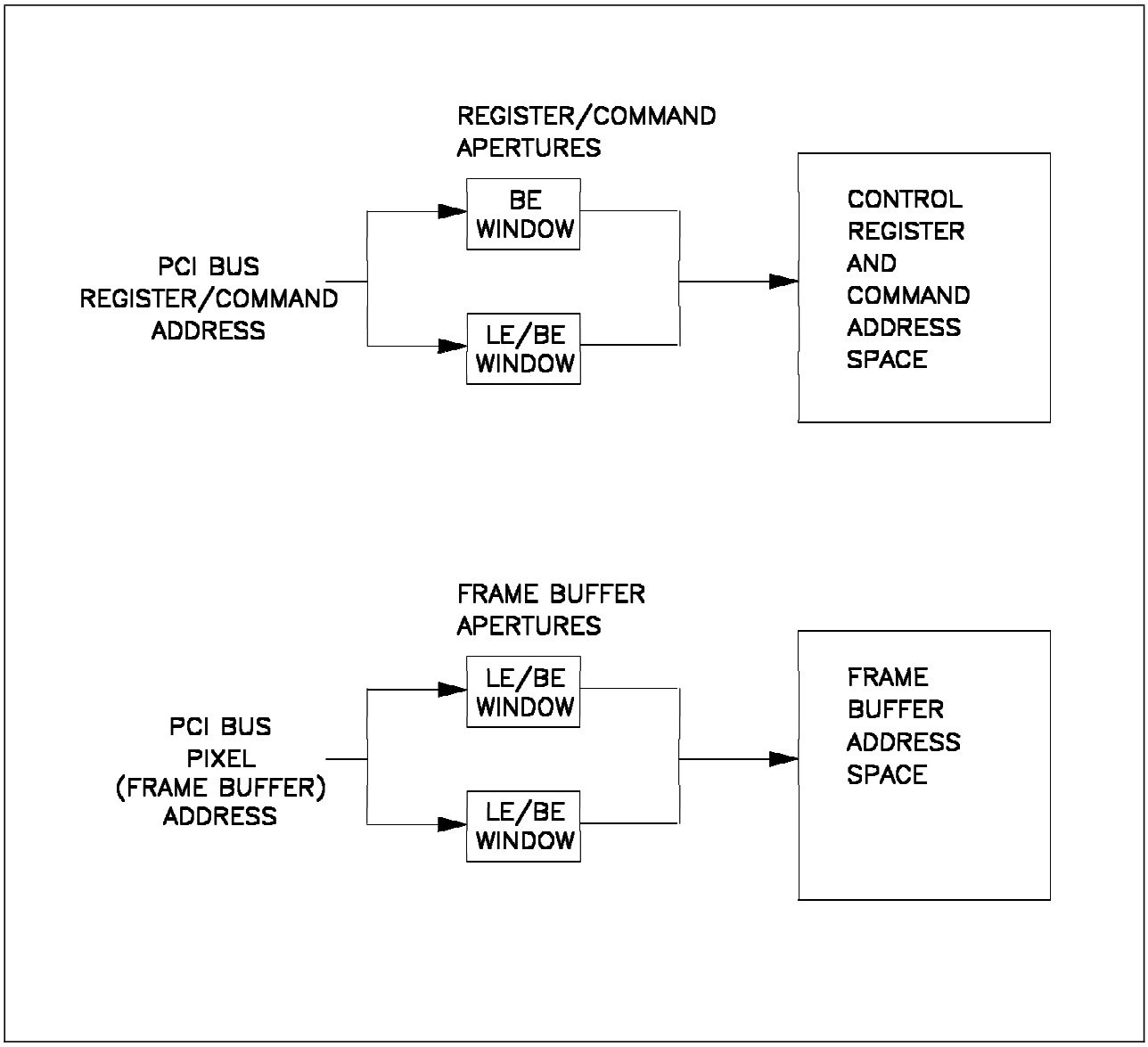


Figure 58. Bi-Endian Apertures for the Graphics Subsystem

formance when using the hardware acceleration features of the graphics subsystem. In this design, the graphics subsystem is accessed through address “apertures” which perform the Endian switch of data as it passes through the aperture.

The register/command apertures provide two ways (Big- or Little-Endian) to access the control data. They are defined address ranges within the address map of the adaptor. While two dedicated apertures are desired, one aperture may be used if it can programmably accommodate Big- or Little-Endian. The aperture labelled “BE” would always provide byte swapping in a two-aperture implementation. The aperture labelled “LE/BE” would provide no byte swapping in a two-aperture implementation, and provide byte swapping in a single-aperture implementation based on one of the following:

a) The size of the facility being accessed would automatically determine how swapping is to occur for a particular access (e.g. half- or full-word, or no swapping).
b) Software would programmably set the swapping function to either halfword, full-word, or none as appropriate.

The frame buffer apertures provide two ways to access the pixel data. The frame buffer apertures are defined address ranges within the address map of the adaptor. Each frame buffer aperture can be independently controlled to provide one of the following modes, under software control:

a) No swapping
b) Byte swapping within each halfword, e.g. A-B-C-D = > B-A-D-C
   - This becomes A-B-C-D-E-F-G-H = > B-A-D-C-F-E-H-G for a system with a 64-bit data interface, such as the PCI bus with optional 64-bit bus extension
c) Byte swapping across the entire word, e.g. A-B-C-D = > D-C-B-A
   - This becomes A-B-C-D-E-F-G-H = > D-C-B-A-H-G-F-E for a system with a 64-bit data interface, such as the PCI bus with optional 64-bit bus extension
d) Byte swapping across a doubleword, e.g. A-B-C-D-E-F-G-H = > H-G-F-E-D-C-B-A, for 64-bit pixels on the 64-bit extended data bus

Note: Determination of the correct swapping mode for a given access may be performed automatically in the hardware. This approach is not recommended if there is a possibility that the pixel depth in the frame buffer can be interpreted differently by multiple devices, or is not guaranteed to be “known” or implied by the state of the graphics subsystem hardware.

With this capability, one aperture could access the frame buffer in Little-Endian mode, while the other could access it in one of the Big-Endian modes. Similarly, one aperture could be defined to swap for 16-bit pixels, / while the other could be defined to swap for 24- or 32-bit pixels. Alternatively, several apertures may be / defined to support the various pixel depths.

Systems employing only one frame buffer aperture would provide the previously described swapping options under software control.

Twenty-four-bit pixels are defined to occupy the least significant 3 bytes of a full word. The most significant byte may be used as an alpha byte, or may be unused. Sixteen-bit pixels are always halfword aligned, and 24- or 32-bit pixels are always full-word aligned.

Pixels between 32 and 64 bits are defined to occupy the least significant bytes in the 64-bit field, and are doubleword aligned.

B.5 Future Directions in Bi-Endian Architecture

This section discusses directions in the design of future PowerPC chips and the effect these changes will have on PowerPC Reference Platform system design. The current implementation of Little-Endian in the PowerPC chips reduces internal processor complexity by moving some of the Bi-Endian support out of the processor. This implementation has two disadvantages:

a) The PowerPC processor chip expects data presented in Big-Endian order.
b) The processor in Little-Endian mode interrupts when it encounters unaligned loads and stores, multiple loads and stores, and string loads and stores.

The first disadvantage impacts cost, complexity, and chip portability. For Little-Endian hardware implementations, the current processor requires byte reversal and address translation hardware external to the processor chip. This extra hardware adds complexity and cost to the machines that are being designed and limits the applicability of the processor chip exclusively to designs that have this logic. Without this restriction, the processor chip could be incorporated into Little-Endian designs without modification to that design.

The second disadvantage is a performance and portability impact. Performance is impacted because if a program uses unaligned loads and stores, load or store multiple instructions, or string operations, then the interrupt handler must emulate these instructions. For proper operation, the interrupt handler must translate

unaligned loads and stores to loads and stores of aligned pieces, and must translate load and store multiple instructions and string operations to loops. The Little-Endian program has a performance impact from the interrupt and the emulation. Alternatively, the Little-Endian source code which is to be transported to a PowerPC environment may be changed to eliminate unaligned mappings, and compilers may be modified to not generate multiple loads and stores or string operations. This requirement impacts portability of application source code and requires different logic in compilers supporting Little-Endian-mode PowerPC implementation.

Future versions of the architecture may permit PowerPC processor chips to support a true implementation of the Little-Endian mode. In this true Little-Endian mode, data would go to the processor in Little-Endian format and be addressed from the processor with the Little-Endian address. Additional hardware support for Little-Endian unaligned operations might be provided.

Figure 59 shows the design to be used with any future version of the PowerPC processor that implements full Bi-Endian support. No address modifications or byte reversal multiplexors are required. The previous designs may be migrated to this design by physically removing the byte reversal multiplexors and address modification logic components from the design, or by functionally removing them by changing the rules under which they are applied. For instance, the Reference Implementation of the Bi-Endian Memory and Bi-Endian Transportable I/O design, Figure 56, would require the address modification and byte-transposing multiplexor to always be off. The address modification is no longer needed, because the processor would not modify the address since it deals with Little-Endian data in Little-Endian order. The byte reversal is not required because the processor accepts data in Little-Endian order.

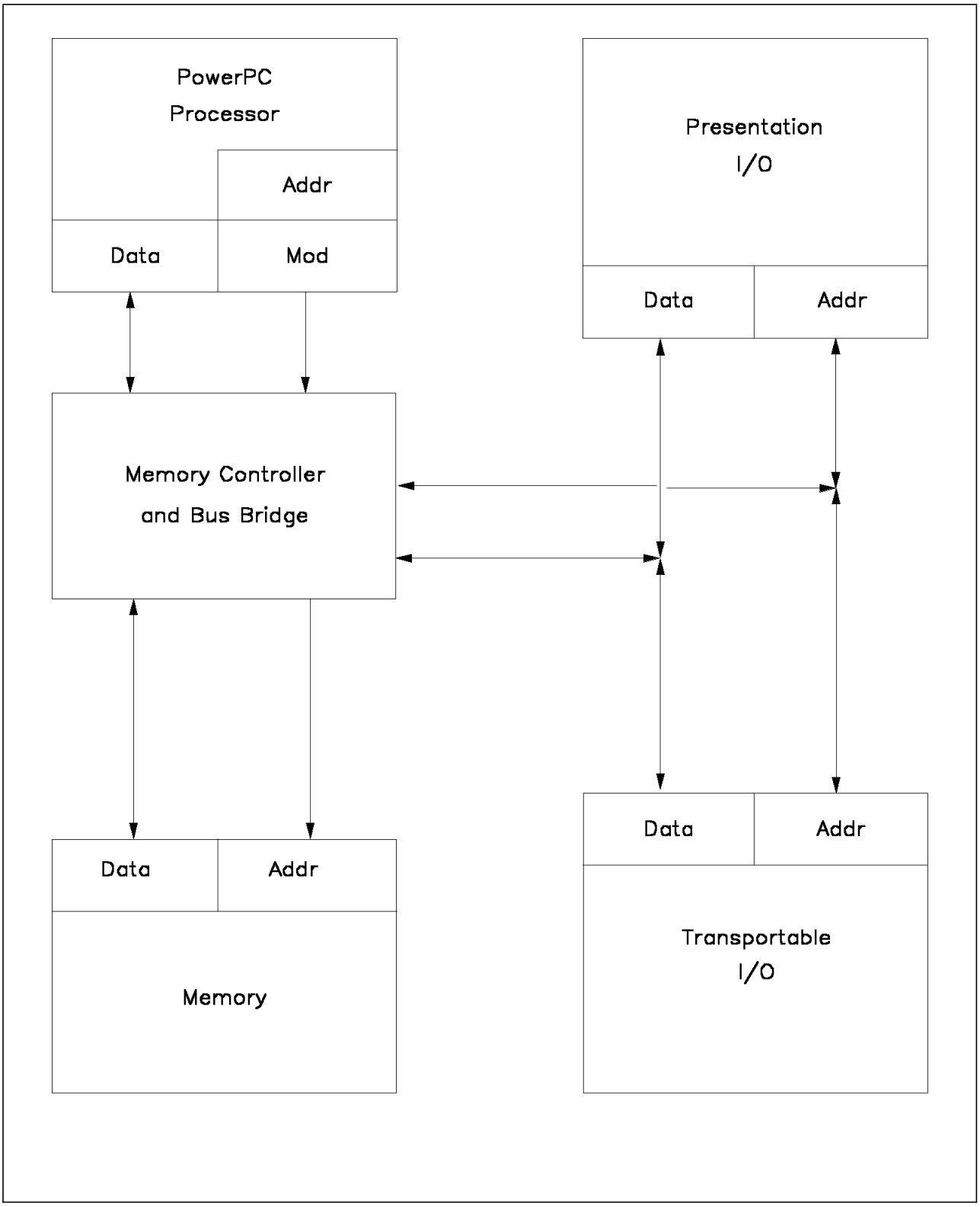


Figure 59. Design with a Full Bi-Endian Processor

Appendix C. Additional Compliant Subsystems and Devices

/ This section lists devices which can be used to configure a PowerPC Reference Implementation. As of the
/ publication date of this specification, this list of adaptors supports this PowerPC Reference Implementation.
/ As additional devices gain software support, they will be added to this list. The last subsection lists specific
/ regional device characteristics which should be considered.

/ C.1 Native Subsystems

/ It is strongly recommended that all operating systems support the items in the following list of directly
/ attached adaptors and devices. For maximum compatibility, system designers should use these devices. The
/ adaptors and devices listed are identical to the same adaptors and devices used in Intel-based PCs. It is
/ expected that the majority of these devices have Intel-based device drivers, simplifying the port to PowerPC
/ Reference Platform systems.

- Keyboards
  - / – Space-saving 84-, 85- and 89-key
  - | – Enhanced keyboard
- Pointing Devices
  - – PS/2 Mouse and compatible
  - | – TrackPoint II and TrackPoint III
- SCSI Adaptors
  - – NCR810/720 (clones may require device drivers for Adaptec and Future Domain adaptors)
- SCSI-Attached Devices:
  - – Hardfiles
    - — All hardfiles compatible with standard SCSI interface; sizes range from 240 MB to 2 GB.
  - – CD-ROMs
    - — Toshiba 4401/3401 and compatible
  - – Tapes
    - — IBM 1.2 GB External
    - — 2.0 GB HP 35470A DAT
    - — HP 35480 DAT
    - — WangDat 3200
    - — Wangtek 5525ES (QIC)
    - — 51000ES (QIC)
    - — Tandberg 3820
    - — Tandberg 4120
  - – Miscellaneous
    - — IBM 3.5" optical
    - — IBM R/W optical
    - — Microarray RAID5 Disk Array
    - — OASIS RAID5 Disk Array
  - – Scanners
    - — HP ScanJet IIp/c and compatible (VEGA standard)
- / IDE
  - / – Opti 82C621 Enhanced IDE

- Floppy Adaptor
  - Intel 82077 and Compatible
- Floppy Drives
  - Non-IBM 720 KB/1.44 KB/2.88 KB 3.5" and 1.2 MB 5.25"
- Audio Adaptor
  - AD1848 (Business Audio Compatible)
  - Crystal Semiconductor CS4231
- Serial Ports
  - 16550 and Compatible
- Parallel Ports
  - Standard Parallel Port, Enhanced Parallel Port, Extended Capabilities Port
- Printers (Only vendors are listed. Epson (dot), HP (PCL), HP (plotter), and Postscript-compatible printers are supported.)
  - Brother
  - Canon
  - Citizen
  - Epson
  - Fujitsu
  - HP
  - IBM
  - Kyocera
  - Lexmark
  - OKI
  - Panasonic
  - QMS
  - Tektronix

C.2 PCI Subsystems

It is strongly recommended that all operating systems support the items in the following list of PCI adaptors and devices. For maximum compatibility, system designers should use these devices. The adaptors and devices listed are identical to the same adaptors and devices used in Intel-based PCs. It is expected that the majority of these devices have Intel-based device drivers, simplifying the port to the PowerPC Reference Platform system.

- Graphics Controllers (1024x768x8, 1024x768x16, 1280x1024x8)
  - S3 928 (used in development) or newer Vision864
  - Weitek 9000 (used in development) or newer 9100 (Bi-Endian capable)
  - IBM PowerGTX 150P (Bi-Endian capable)
  - Western Digital 90C24A2 as used in the Portable

C.3 PCMCIA Subsystems

It is strongly recommended that all operating systems support the PCMCIA adaptors listed below. For maximum compatibility, system designers should use these devices. These PCMCIA adaptors are identical to the ones used in Intel-based PC systems. The adaptors are also similar to their ISA-based counterparts. The main support required is PCMCIA socket services. The majority of the following adaptors are for communications.

- Ethernet
  - IBM PCMCIA Ethernet Card
  - Xircom PCMCIA Ethernet Card
- Modems (Data/Fax)
  - AT&T Paradyne 14.4/9.6 PCMCIA
  - Hayes 2.4/9.6 PCMCIA
  - IBM PCMCIA Modems
  - Megahertz 14.4/14.4 PCMCIA
  - Megahertz 2.4/9.6 PCMCIA
- Hardfiles
  - Maxtor 105-MB Type 3 PCMCIA
  - Sundisk Silicon Hardfiles
- Token Ring
  - IBM PCMCIA Token Ring Card
- Audio
  - IBM MWAVE PCMCIA Audio Cards
- Pager
  - Motorola Newscard**
- ISA-to-PCMCIA Bridge
  - Intel 82365SL
  - Ricoh RF5C266
  - Ricoh RF5C366
  - Cirrus CL_PD6710
  - Cirrus CL_PD6720
  - IBM CTLISA2 (2 ports)

C.4 ISA Subsystems

It is strongly recommended that all operating systems support the items in the following list of ISA adaptors and devices. For maximum compatibility, system designers should use these devices. The adaptors and devices listed are identical to the same adaptors and devices used in Intel-based PCs. It is expected that the majority of these devices have Intel-based device drivers, simplifying the port to the PowerPC Reference Platform system.

- Network Adaptors
  - 3Com Etherlink ISA Adaptors
  - Intel EtherExpress 16
  - Novell NE2000/3200 (ISA)

  - Ungermann-Bass NIUPS Adaptors
  - Standard Microsystems Ethercard ISA Adaptors
  - IBM Token Ring or Ethernet Adaptors

- Fax/Data Modems (most are Hayes compatible and use a serial port interface)
  - Hayes Modems (2400 - 14400) and Compatible
  - Intel SatisFAXtion Modems
  - Megahertz Modems
  - Practical Peripherals Modems
  - US Robotics Modems

- Other Adaptors
  - IBM M-Wave Products (ISA)
  - Creative Labs Sound Blaster Pro/16 (ISA)

C.5 Regional Device Characteristics

Within some countries or regions, common usage suggests that additional device characteristics be considered for PowerPC Reference Platform systems. These considerations are listed below:

Country	Suggested device characteristics
Japan	The 3.5-inch floppy drives should be capable of 1.2 MB as well as the 1.44-MB format. To support this 1.2-MB format, the drive may have to support a 360-rpm speed in addition to the 300-rpm speed.
Europe	Ergonomics standards may affect the system design.

Appendix D. Windows NT

This appendix describes the Windows NT operating system which will run on PowerPC Reference Platform-compliant machines.

D.1 Operating System Scope

/ The Windows NT operating system supports portable, desktop and server environments. The operating
| system has dynamically loadable software device drivers and a software layer that can be used for hardware
| adaptation. This Hardware Abstraction Layer (HAL) and the device drivers allow hardware vendors some
/ freedom to differentiate.

/ A Windows NT porting center is available to help vendors who are interested in porting their products to
/ this platform. The center can be contacted by telephone at 1-800-803-0110 or 1-206-889-9011, or by elec-
/ tronic mail on Internet at winntppc@vnet.ibm.com.

D.2 Operating System Version

/ The current version of Windows NT will support this implementation.

/ D.3 Operating System Environment

/ Windows NT runs in the Little-Endian mode.

D.4 Operating System Configuration

/ For the purpose of describing configurations which support Windows NT, three configurations are listed
/ below:

Configuration Name	Configuration Definition
Client Workstation	A basic single-user system which operates either stand-alone or as a client in a network. Software includes the base system, utilities, file systems, shared printers, performance/event monitoring, backup, remote access, network client support, system management and several personal productivity aids such as mail and workgroup scheduling.
Developer Workstation	The developer workstation is essentially the same as the client workstation. A developer might need additional disk space and would purchase compilers and debuggers to assist in application development.
Server	A configuration which provides shared system management such as workgroup-wide passwords and protection attributes. It also provides RAID and disk mirroring support.

D.5 Hardware Configuration Requirements

This section defines the minimum, optional and alternative hardware configuration requirements for three Windows NT operating system configurations. In many cases operating performance improvements can be realized by configuring systems with larger or faster components. Because these configuration adjustments are very application load dependent, no attempt has been made to recommend operationally tuned configurations.

The subsections below describe the configuration alternatives for each of the three configurations. This material is summarized in Table 29.

D.5.1 Processor Subsystem

/ A PowerPC Reference Platform-compliant system must have a PowerPC compliant processor. Windows
/ NT will support PowerPC 601, 603, 604 or compatible processors running at supported frequencies. Performance will be better with higher frequencies and with PowerPC processors having performance-enhancing features (i.e. larger internal cache).

D.5.2 Memory Subsystem

System Memory for the client workstation should be at least 16 MB. For the developer workstation, System Memory should be 24 MB. For a server, System Memory should be 32 MB. Performance improvements can be realized in all three configurations with additional System Memory.

Boot memory will not vary between configurations. The minimum size is determined by what a vendor needs to boot the machine into the state expected by the operating system as defined in Section 5.0, “Boot Process and Firmware.”

An additional cache external to the processor is optional on a PowerPC Reference Platform-compliant system. Minimum configurations for all three workstation configurations do not require an L2 cache. For performance reasons, the developer workstation could have a 256-KB L2 cache and the server could have a 512-KB L2 Cache.

Non-volatile Memory to support Windows NT will not vary between workstation configurations.

D.5.3 Storage Subsystems

The minimum hardfile size for a client workstation is 200 MB. A developer workstation would require at least 400 MB. The minimum size for a server depends upon its function. A simple print server with only one printer could be 400 MB. A data server would need significantly more hardfile space in the range of 2 to 4 GB.

A single 1.44-MB 3.5-inch floppy drive should be the minimum configuration for all three workstation types. Windows NT also supports the 2.88-MB 3.5-inch floppy drive format.

A CD-ROM is required in the minimum configuration for all Windows NT machines. Windows NT will only be shipped on this media. Product manuals, information, and command references are available through this media. An acceptable alternative to locally attached CD-ROM is network access to material on CD-ROM. The developer workstation and the server workstations could benefit from CD-ROMs with high access speeds.

D.5.4 Human Interface Subsystem

All configurations must have a graphics resolution of at least 640 by 480 pixels.

/ A keyboard and pointing device are required on all configurations.

A business audio device is required.

All configurations need the standard Real-Time Clock.

D.5.5 Connectivity Subsystem

/ Windows NT does not require a serial port, but a serial port is required as part of the standard hardware
/ configuration described in Section 2.0, “Hardware Configuration.” Windows NT supports EIA/TIA-232E-compliant serial ports. Any configuration may optionally add one or more serial ports.

None of the three configurations require a parallel port. Windows NT supports 8-bit bidirectional Compatibility Mode-compliant parallel ports.

No network connection is required. Both Ethernet and Token Ring are supported, as is full remote access over phone lines.

D.5.6 Expansion Bus Interfaces

/ Windows NT supports a PCI secondary bus. Additional parallel PCI buses or different secondary buses
/ could be used, but the vendor would have to supply modified abstraction software.

The ISA bus is optional for all three configurations. As I/O adaptors become readily available on the PCI bus (or alternative secondary bus), configurations may remove the ISA bus.

Windows NT will support PCMCIA, but a vendor including this connector must supply a compatible device driver.

Windows NT supports multiple SCSI interfaces.

Windows NT supports IDE access to disks.

/ D.5.7 Security and System Management

/ Windows NT does not require special features for security and system management.

D.6 Hardware Configuration Recommendations

The Windows NT operating system which supports PowerPC Reference Platform-compliant systems will support additional equipment. In some cases, drivers will have to be supplied by either the system vendor or the device vendor.

/ Tape drives are useful additions to PowerPC Reference Platform-compliant systems as backup and archive
/ devices. Windows NT supports three formats: Quarter Inch Cartridge (QIC), 4 mm, and 8 mm.

Multimedia upgrades with better sound and MIDI support would be useful additions to a system. The vendor would have to provide NT-compatible drivers.

Alternative graphics adaptors may be supplied on PowerPC Reference Platform-compliant machines.
/ Windows NT currently supports S3, Weitek, and Western Digital. Vendors would have to provide drivers for other adaptors.

D.7 Boot Time Abstraction Requirements

/ Windows NT uses the boot devices defined by the firmware. Only those devices (e.g. video, keyboard, disk)
/ known to the firmware participate in the boot process.

D.8 Hardware Abstraction Layer

/ The following table indicates the effect that changes in hardware features may have on Windows NT. The
/ left column lists those features of the hardware that may impact the operating system if changed. Some
/ features listed do not cause a system change. These are features that one would expect to cause operating
/ system changes, but in the case of NT, do not. The next four columns indicate which major operating
/ system components are affected by the feature change. *Firmware* refers to the ARC firmware that provides
| hardware-specific function to the hardware-independent OS Loader. *HAL* refers to the software layer that
| provides the hardware adaptation function. Firmware and HAL can be changed by hardware system
/ vendors and shipped with their products. Kernel changes are made only by Microsoft and Windows NT
/ source licensers such as IBM and Motorola. Device drivers can be distributed between Microsoft releases via
/ CompuServe and bulletin boards as well as with hardware.

/ An "X" in a box indicates that changing the hardware feature causes changes in the indicated operating
/ system component. An "[X]" indicates that the current beta version of the PowerPC port requires a change
/ but may not require change in the final release. The Currently Supported column lists the current parts that
/ are supported with the existing software.

/ This chart indicates the need to look for potential consequences of feature changes. The specific effects of
/ any hardware change must be evaluated to determine the extent of software modification required. For
/ instance, changing the RTC is a straightforward process, while changing the coherency protocol for the cache
/ could be a major effort.

/ Hardware system vendors who require firmware changes will need to obtain a Firmware Kit and the appro-
/ priate licenses from IBM or Motorola. A license may also be required from Microsoft. Hardware system
/ vendors will be notified of this requirement when they request the Firmware Kit. For HAL changes, hard-
/ ware system vendors must obtain a HAL license from Microsoft and a HAL kit from IBM or Motorola.
/ For device driver changes, a Device Driver Kit from IBM or Motorola is required.

/ Table 28 (Page 1 of 2). Effect of Hardware Changes on Windows NT

/ /	**Component**	**Firmware**	**Kernel**	**HAL**	**Device Driver**	**Currently Supported**
/	PROCESSOR:					601, 603, 604
/	Numbers of Processors	X	X	X		
/	Speed					
/	Cache Organization	X		X		
/	Cache Size	X		[X]		
/	SPRs	X	X	X		
/	TLB		X			

Table 28 (Page 2 of 2). Effect of Hardware Changes on Windows NT

Component	Firmware	Kernel	HAL	Device Driver	Currently Supported
Memory Management		X			
Processor clock			X		
Exception Vector	X	X	X		
L2 CACHE:					
Size	X		X		256 KB
Coherency	X	X	X		
MEMORY CONTROLLER:					IBM 27 82650
Speed					
Interrupt Acknowledge	X		X		
PCI Configuration Interface	X		X		
Memory Timing	X				
BUS BRIDGE CONTROLLER:					Intel SIO 82378IB/ZB
Speed	X				
Interrupt Controller	X		X		8259
Timer			X		8254
DMA Controller			X		
SUPER I/O	X		X		SMC FDC37C665, National PC87312
REAL-TIME CLOCK	[X]		X		Dallas 1385S, 1585S
POWER MANAGEMENT	X		X		
NVRAM	X		X		Dallas 1385S, 1585S
SCSI SUBSYSTEM	X			X	NCR 810
GRAPHICS SUBSYSTEM	X		[X]	X	S3 928 & 864, Weitek 9000 & 9100, WDC 90C24A2
KEYBOARD CONTROLLER	X			X	i8042
PERIPHERAL BUS:					PCI/ISA
Type	X		X		
Number			X		
Peripheral Devices				X	

D.9 Device Driver Model

There is a separate Device Driver Development Kit (DDK) available from both Microsoft and Motorola. It contains all the documentation and sample code needed to develop device drivers.

D.10 Multiprocessing Model

Windows NT supports SMP.

D.11 Application Model

Reference the ABI/API document for Windows NT.

D.12 Configuration Summary Table

Table 29 gives the minimum configuration information for each of the three Windows NT system configurations. This table specifies levels of hardware necessary to run the three Windows NT system configurations. In many cases upgrade options are possible. Some of those options and configuration alternatives were described in Section D.5, “Hardware Configuration Requirements.”

Table 29 (Page 1 of 2). Hardware Requirements for Windows NT System Configurations

	System Component	Client	Developer	Server
	Processor	PowerPC 601, 603, 604 or compatible processor		
	System Memory	16 MB	24 MB	32 MB
	Boot Memory	Standard	Standard	Standard
	Non-volatile RAM	Standard 4 KB	Standard 4 KB	Standard 4 KB
	L2 Cache Memory	Optional	Optional	Optional
	Hardfile	200 MB	400 MB	400 MB; 2 GB
/	Floppy	Optional	Optional	Optional
/	CD-ROM*	Required	Required	Required
/	Alphanumeric Input Device	Required	Required	Required
/	Pointing Device	Required	Required	Required
/	Audio	Standard	Standard	Standard
/	Graphics	640x480	640x480	640x480
	Real-Time Clock	Standard	Standard	Standard
/	Serial Ports	Standard	Standard	Standard
	EIA/TIA-232E	Windows NT supports this serial interface		
	EIA-422	Windows NT does not support this serial interface		
	Parallel Ports	Optional	Optional	Optional
	Compatibility Mode	Windows NT supports this parallel interface		

Table 29 (Page 2 of 2). Hardware Requirements for Windows NT System Configurations

System Component	Client	Developer	Server
Extended Capabilities Port	Windows NT does not support this Parallel interface		
Network Adaptors	Optional	Optional	Required
Ethernet	Windows NT supports Ethernet		
Token Ring	Windows NT supports Token Ring		
SCSI	Optional	Optional	Optional
IDE	Optional	Optional	Optional
PCI Bus	1 Required	1 Required	1 Required
ISA Bus	Optional	Optional	Optional
PCMCIA	Optional -- but a vendor-supplied device driver is required		
Tape Drive	Optional -- but a vendor-supplied device driver is required		

Legend:

Entry	Definition
Standard	The hardware configuration in Section 2.0, “Hardware Configuration,” requires this component.
Required	This component must be present in the system on which Windows NT will run.
Optional	This component is supported by Windows NT, but is not required.
*****	Capability can be provided via a network.

Appendix E. AIX

This appendix provides an overview of the AIX (R) Operating System which will run on PowerPC Reference Platform-compliant machines.

E.1 Operating System Scope

AIX, IBM's implementation of the UNIX** operating system, offers scalability from entry-level systems to symmetric multi-processing (SMP) servers suitable for enterprise mission-critical applications. The combination of networking capabilities, graphics, usability, performance, packaging, and standards compliance built into AIX make it an attractive operating environment in standalone configurations or as a client or server in networked environments.

AIX has dynamically loadable software device drivers and is modularized to assist in hardware adaption, allowing vendors the freedom to differentiate their system offerings.

AIX Version 4.1, announced July 26, 1994 and refreshed with AIX Version 4.1.1 on October 4, 1994, is the most significant enhancement to AIX since its initial introduction. AIX Version 4.1.1 features:

- SMP support
- Installation packages tailored for client and server configurations
- Improvements such as greater-than-2-GB file systems, AIX kernel threads, and enhancements to the install process
- Journaled File System (JFS) Support for disk fragmentation and dynamic compression/decompression
- A new graphical user interface (GUI) based on the Common Desktop Environment
- Improved standards alignment for compatibility with other open systems, including the X/Open XPG4 Base Profile, PowerOpen and Spec 1170
- Improved ease of use with the addition of a GUI for installation and automatic installation of device drivers

For AIX license and product information, contact David Hall, AIX OEM Relations, at 512-838-2088.

Software vendors interested in porting their applications to AIX on PowerPC systems can contact the AIX Power Team General Information Line at 1-800-222-2363.

E.2 Operating System Version

This appendix describes AIX Version 4.1.1.

E.3 Operating System Environment

AIX runs in the Big-Endian mode. AIX supports the XCOFF object module format as defined in the *PowerOpen ABI*.

E.4 Operating System Configuration

/ AIX supports a range of system configurations. For the purpose of defining configuration size, three are
/ listed below:

/ Configuration Name	Configuration Definition
/ **Client Workstation**	A basic single-user system which operates either stand-alone or as a client in a / network.
/ **Developer Workstation**	A client configuration plus additional utilities, compilers and debuggers to assist in / application development.
/ **Workgroup Server**	An entry-level configuration which provides shared resources such as a printer or / file storage to a local work group over a Local Area Network. This configuration / includes the developer workstation functions plus server support and networking / support.

E.5 Hardware Configuration Requirements

This section defines the minimum, optional and alternative hardware configuration requirements for the three configurations. In most cases, operating performance improvements can be realized by configuring systems with larger or faster components. Because these configuration adjustments are very application load dependent, no attempt has been made to recommend operationally tuned configurations.

The subsections below describe the configuration alternatives for each of the three configurations. This material is summarized in Table 31.

E.5.1 Processor Subsystem

/ A PowerPC Reference Platform-compliant system must have a PowerPC compliant processor. AIX will
/ support PowerPC 601, 603, 604, or compatible PowerPC processors running at any supported frequency. Performance will be better with higher frequencies and with PowerPC processors having performance-enhancing features (i.e. larger internal cache).

E.5.2 Memory Subsystem

System Memory for the client workstation should be at least 16 MB. For the developer workstation, System Memory should be 24 MB. For a LAN server, System Memory should be 32 MB. System Memory should start at address zero and should be continuously populated through the maximum amount in the configuration (e.g. no holes). Performance improvements can be realized in all three configurations with additional System Memory.

System ROM will not vary between configurations. The minimum size is determined by what a vendor needs to boot the machine into the state expected by the operating system as defined in Section 5.0, “Boot Process and Firmware.”

A cache external to the processor is optional on a PowerPC Reference Platform-compliant system. Minimum configurations for all three workstation configurations do not require an L2 cache. For performance reasons the developer workstation could have a 256-KB L2 cache and the LAN server could have a 512-KB L2 Cache.

/ Non-volatile Memory to support AIX will not vary between workstation configurations. AIX conforms to
/ the mapping of NVRAM provided in this document.

E.5.3 Storage Subsystems

The minimum hardfile size for a client workstation is 200 MB. A developer workstation would require at least 400 MB. The minimum size for a LAN server depends upon its function. A simple print server with only one printer could be 400 MB. A data server would need significantly more hardfile space in the range of 2 to 4 GB.

A single 1.44-MB 3.5-inch floppy drive should be the minimum configuration for all three workstation types. AIX also supports the 2.88-MB 3.5-inch floppy drive format.

A CD-ROM is required in the minimum configuration for all AIX machines. AIX will only be shipped on this media. Product manuals, information, and command references are available through this media. An acceptable alternative to locally attached CD-ROM is network access to the material which is transmitted on CD-ROM. This alternative requires that the system boot from a network because AIX does not support installation and maintenance from a floppy. The developer workstation and the LAN server workstations could benefit from CD-ROMs with high access speeds.

E.5.4 Human Interface Subsystem

All configurations should have an alphanumeric input device. If the LAN server does not function also as a developer or client workstation, they may use a simple console (e.g tty).

A pointing device is required on the client workstation and the developer workstation. If the LAN server does not function as one of these types of workstations in addition to being a server, then it does not need a pointing device. Normally a mouse is used as the pointing device. The minimum requirement is a two-button mouse. A three-button mouse is strongly recommended because X Windows on AIX does not
/ support a two-button mouse. Alternatively, the Track Point II or Track Point III may be used.

/ A business audio device is required as part of the standard hardware configuration described in Section 2.0,
/ “Hardware Configuration,” but is not required by AIX on any workstation. A PC-type speaker may be used for error and warning sounds.

A graphics system capable of at least 800x600 pels is necessary for the client workstation. A graphics system capable of at least 1024x768 pels is the minimum configuration recommended for a developer workstation. A LAN server may have only an ASCII character video system unless it also performs the functions of a client or developer workstation. The maximum resolution currently supported by AIX is 1280x1024 pels.
/ For the best graphics performance, a Bi-Endian graphics adaptor should be provided.

All configurations need the standard DS 1385S Real-Time Clock. Current implementations depend upon this specific clock chip.

E.5.5 Connectivity Subsystem

/ All configurations require the two EIA/TIA-232E-compliant serial ports which are natively attached to the
/ system as in the Reference Implementation. The AIX kernel debugger requires a serial port when it is oper-
/ ating. AIX is not distributed with support for EIA-422. System vendors who desire this capability will have to write a device driver to support it. Any configuration may optionally add more serial ports, but device drivers will have to be supplied by the vendor.

/ All three configurations require a parallel port which is natively attached to the system as in the Reference
/ Implementation. AIX supports 8-bit bidirectional Compatibility Mode-compliant parallel ports. AIX does

not support the Extended Capabilities Port, IEEE P1284. Vendors who want to include this interface must provide a device driver. Any configuration may optionally add more parallel ports.

/ All three configurations may have optional network interfaces of either Ethernet or Token Ring.

E.5.6 Expansion Bus Interfaces

All three configurations require one and only one PCI bus.

/ The ISA bus is required for all three configurations. As I/O adaptors become readily available on the PCI bus, configurations may remove the ISA slots. However, the ISA bus decoder is required for native I/O support for such interfaces as parallel, serial, keyboard, and mouse.

AIX does not currently support PCMCIA. A vendor including this connector must supply a compatible device driver and operating system extensions.

/ AIX supports the natively attached SCSI interface as in the Reference Implementation. One or more
/ hardfiles may be attached to this interface.

/ AIX does not support IDE disks.

/ E.5.7 Security and System Management

/ AIX requires no special features for security and system management.

E.6 Hardware Configuration Recommendations

The AIX system which supports PowerPC Reference Platform-compliant systems will support additional equipment. In some cases, drivers will have to be supplied by either the system vendor or the device vendor.

/ Tape drives are useful additions to PowerPC Reference Platform-compliant systems for backup and archive.
/ AIX supports SCSI-attached tape drives in three formats: Quarter Inch Cartridge (QIC), 4 mm, and 8 mm.

Multimedia upgrades with better sound and MIDI support would be useful additions to a system. The vendor would have to provide AIX-compatible drivers.

Alternative graphics adaptors may be supplied on PowerPC Reference Platform-compliant machines. AIX
/ currently supports S3 928 or Vision 864, Weitek 9000 or 9100, or IBM PowerGTX 150P. Hardware
/ vendors may supply alternate adaptors provided that the adaptors work on PCI buses, have low-level and X
/ Windows drivers compatible with AIX, and are supported by the boot process.

E.7 Boot Time Abstraction Requirements

Certain memory map requirements must be satisfied for AIX to be booted and to operate. Low memory from X'00000000' through X'00004FFF' must be reserved for system use. Table 30 shows the allocation of this real memory space. The AIX boot process requires that the space from this point through 2 MB be available and reserved for the kernel until the virtual to real address translation is established. After that, for the remainder of the boot and for subsequent operation of AIX, real memory up to 0.5 MB must be available and dedicated to the AIX kernel. These additional restrictions due to AIX place some limits upon a vendor's freedom in assigning memory map addresses.

Table 30. Low Memory Area Definition	
Location	**Description**
X'00000000'-X'2FFF'	PowerPC architected interrupt vector location. See *The PowerPC Architecture*, Chapter 13.
X'00003000'-X'30FF'	IBM copyright notice
X'00003100'-X'3FFF'	Usermode milicode routines and user mode kernel routines
X'00004000'-X'47FF'	Kernel patch area
X'00004800'-X'4FFF'	Kernel overlay area and kernel branch table

E.8 Hardware Abstraction Layer

/ An abstraction layer is, fundamentally, a way of structuring the operating system environment to isolate its hardware-dependent components and facilitate the introduction of changes in the processor architecture and system structure. It is an essential component of how the kernel, subsystems, device drivers, and, ultimately, applications can be made to run compatibly across the product line and from release to release. It can be used very effectively to assist in the porting of an operating system to variations of existing platforms.

/ The AIX kernel has a modular structure and a formalized and documented set of interfaces. These attributes
/ will persist across operating system releases and can achieve many of the same goals as the abstraction soft-
/ ware. Vendors who desire to differentiate their platforms for AIX must obtain a source license and then
/ modify kernel components which are affected by the hardware differences.

The introduction of any layer between the hardware and software inevitably raises the question of performance impacts. In the AIX kernel definition, we allow the performance optimizations to occur in a well-defined manner, and it is our intent to provide extensions to the interface to meet future requirements. The AIX kernel, as it is defined, enhances the portability of AIX, as we continue to implement our palmtop-to-teraflop platform vision. There are a number of other related benefits of the structured AIX kernel, such as resource utilization, product time-to-market, and maintenance and support, which are not discussed here.

From a structural view, the routines in the kernel which interface to hardware components will be provided with a set of services to perform a requested function independent of the underlying hardware. The main components of the AIX kernel fall into categories of service which account for differences in processor, I/O, and platform-specific implementation. Typical services which will be provided under this framework are memory management services (cache and DMA), I/O services (access to bus controllers and system I/O), bring-up and configuration services (hardware initialization), interrupts, device drivers, and RAS (access to NVRAM and system error registers). As an example of one of these services, in the hardware initialization routine for an operating system supporting platforms with different I/O models, static or dynamic checking of each implementation would have to be included in the routine. In the AIX environment, the routine would make a call to a kernel component that is platform specific.

/ Figure 60 conceptually shows the AIX kernel framework. The kernel hardware services make the underlying hardware accessible to the kernel, device drivers, and subsystems in an abstract manner. The service which is invoked by the routines within the system components remains the same for any number of implementation variations, and the specific processor architecture (Power or PowerPC), platform (PowerPC Reference Platform, PowerPC Reference Platform-clone or SMP), or I/O (PCI or ISA) variation would be accounted for
/ only once, in this service layer. As is shown by the links to these hardware services, there are services defined for the kernel, such as initialization and configuration; others which may be subsystem or device driver specific, such as DMA; and those that may apply to any, such as cache management.

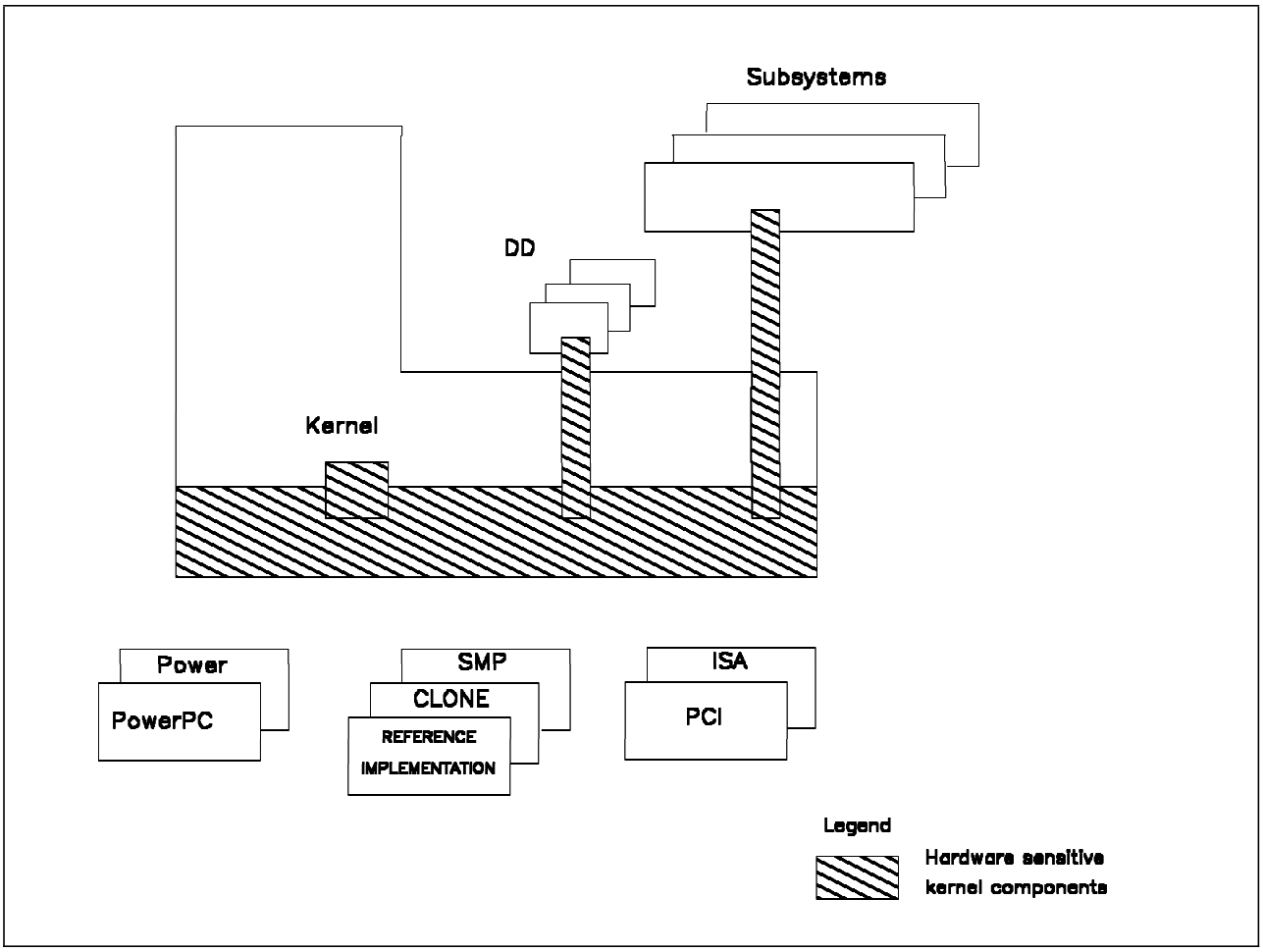


Figure 60. Structure of the AIX Kernel

E.9 Device Driver Model

A device driver is a section of code that provides support for a device. Device drivers run in a privileged state, as AIX kernel extensions, and have access to a number of functions that are unavailable to normal application programs. They shield the user from device-specific details and provide a common I/O model for accessing the devices for which they provide support.

Device drivers can play two roles in AIX: the “device head” role and the “device handler” role.

A device head is a device driver or a portion thereof that provides interfaces to application programs via the standard *open, close, read, write,* and related system calls. A device driver acting in this role takes I/O requests from application programs and communicates them to a device handler. The interface between application programs and a device head is rigidly defined by the AIX kernel itself.

A device handler is the portion of a device driver that communicates with the actual device or adaptor. The device handler takes requests from a device head and implements the request on real hardware. The interface between a device head and a device handler is essentially undefined by AIX, though a large number of primitive functions are provided by AIX to assist in constructing the interface. The details, however, are left to the device driver author. The interface between the device handler and the device itself is naturally dependent on the hardware being manipulated, though AIX again provides a set of functions which assist in performing the hardware interface.

Most simple device drivers will in fact act in both roles, but other configurations are possible. Vendors who offer a system with components different from those in the Reference Implementation must support this differentiation with a device driver. In simple cases where the device head and the device handler are in the same device driver, the system vendor would supply both with the system. In another case where the device head and device handler are in different device drivers, the system vendor would only need to match the interface to the device head and supply a device handler.

Documents useful to vendors needing to develop device drivers are as follows:

- *Kernel Extensions and Device Support Programming Concepts*, IBM document number SC23-2207
- *Writing a Device Driver for AIX Version 3.2*, IBM document number GG24-3629

E.10 Multiprocessing Model

AIX Version 4.1.1 supports SMP. However, there is no multiprocessor support in AIX for systems based on the PowerPC Reference Implementation as defined in Section 6.0, “Reference Implementation.”

E.11 Application Model

AIX Version 4.1.1 will support the PowerOpen Environment (POE). This environment provides an application interface which is targeted to be platform and I/O independent. The goal of the POE Application Binary Interface (ABI) is to free software developers from having to consider platform-specific functions and I/O bus dependencies. The ABI is designed to help enable software vendors to produce shrink-wrapped software. The ABI contains the Applications Programming Interface (API), which provides access to operating system libraries and services.

The PowerOpen ABI defines a system interface for compiled application programs. The purpose of this approach is to standardize the binary interface for application programs or some other operating system that complies with the POE specification.

The PowerOpen ABI execution environment contains information that must be provided by a PowerOpen operating system and is available to a PowerOpen application program. The ABI defines a binary interface for application programs that are compiled and packaged for POE implementations on similar hardware architectures. The binary specification includes information specific to the PowerPC computer processor architecture.

An implementation conforming to this standard must meet the following criteria:

- The system shall support all the PowerOpen execution environments, interfaces and headers defined and listed within the ABI specification, which is part of the POE.
- The system may provide additional or enhanced interfaces, headers, and facilities not required by the ABI specification, provided that such additions or enhancements do not affect the behavior of an application that requires only the facilities described in the ABI specification.

Adherence to the PowerOpen ABI guarantees application portability to future versions of an ABI-conformant system and to future PowerPC architecture implementations. This portability is guaranteed at the following levels, depending on the application’s origin, as follows:

ABI-conforming system A computer system that provides all the binary system interfaces for application programs described in the ABI specification and the PowerOpen API.

ABI-conforming program A program written to include only the following system routines:

Commands and other resources included in the ABI.

Programs compiled into an executable file that has the formats and characteristics specified for such files in the chapter on XCOFF in the ABI specification.

Programs whose behavior complies with the rules given in this ABI specification.

A program must not bind in the routines and data structures from the system shared libraries defined in the chapter on libraries in the ABI specification. A program cannot have the routines defined in the shared libraries statically bound into the program.

Binary compatibility Presents a load-and-go environment. Only the physical availability of the application is needed. Applications adhering to this level of compatibility can be moved across compliant systems. Note that the static linking of shared libraries cannot be guaranteed to work and must therefore be avoided.

E.12 Configuration Summary Table

Table 31 gives the minimum configuration information for each of the three AIX system configurations. This table specifies levels of hardware necessary to run the three AIX system configurations. In many cases upgrade options are possible. Some of those options and configuration alternatives were described in Section E.5, "Hardware Configuration Requirements."

Table 31 (Page 1 of 2). Hardware Requirements for AIX System Configurations

	System Component	Client	Developer	LAN Server
	Processor	PowerPC 601, 603, 604, or compatible processor		
	System Memory	16 MB	24 MB	32 MB
	System ROM	Standard	Standard	Standard
	Non-volatile RAM	Standard -- 4 KB	Standard -- 4 KB	Standard -- 4 KB
	L2 Cache Memory	Optional	Optional	Optional
	Hardfile	200 MB	400 MB	> = 400 MB
/	Floppy	Optional	Optional	Optional
/	CD-ROM*	Required	Required	Required
/	Alphanumeric Input Device	Required	Required	Required
/	Pointing Device	Required	Required	Optional
/	Audio	Standard	Standard	Standard
	Graphics	800x600	1024x768	ASCII Characters
	Real-Time Clock	Standard	Standard	Standard
/	Serial Ports	2 Required	2 Required	2 Required
	EIA/TIA-232E	AIX supports this serial interface		
	EIA-422	AIX does not support this serial interface		
/	Parallel Ports	1 Required	1 Required	1 Required

Table 31 (Page 2 of 2). Hardware Requirements for AIX System Configurations

System Component	Client	Developer	LAN Server
Compatibility Mode	AIX supports this parallel interface		
Extended Capabilities Port	AIX does not support this parallel interface		
Network Adaptors	Optional	Optional	Required
Ethernet	AIX supports Ethernet		
Token Ring	AIX supports Token Ring		
SCSI	1 Required	1 Required	1 Required
IDE	Not supported	Not supported	Not supported
PCI Bus	1 Required	1 Required	1 Required
ISA Bus	Required	Required	Required
PCMCIA	Not currently supported -- a vendor-supplied device driver and kernel extensions are required		
Tape Drive	Optional	Optional	Optional

Legend:

Entry	Definition
Standard	The hardware configuration in Section 2.0, “Hardware Configuration” requires this component.
Required	This component must be present in systems on which AIX will run.
Optional	This component is supported by AIX, but is not required.
*****	Capability can be provided via a network.

Appendix F. Workplace OS

/ This appendix describes OS/2 for the PowerPC, which is the first member of the Workplace OS* family of
/ operating systems. OS/2 for the PowerPC runs on PowerPC Reference Platform-compliant machines.

F.1 Operating System Scope

/ Workplace OS defines a family of general-user operating systems which consist of the IBM microkernel, Per-
/ sonality Neutral Services, and multiple personalities. The first member of the Workplace OS family is OS/2
/ for the PowerPC, which runs on PowerPC Reference Platform-compliant machines. The available personal-
/ ities are OS/2* and MVM (DOS personality). The user sees this as a portable OS/2. This document
/ describes the first implementation of Workplace OS and thus the terms Workplace OS and OS/2 for the
/ PowerPC can be used interchangeably. Future members of the Workplace OS family may have slightly
/ different configurations and requirements. Future personalities include UNIX**.

F.2 Operating System Version

This Appendix describes the first release of Workplace OS. This version provides to users and applications the services that they are currently using in OS/2 2.1.

/ F.3 Operating System Environment

/ OS/2 for the PowerPC runs in Little-Endian mode. OS/2 for the PowerPC supports the ELF object
/ module format with Workplace OS extensions. The linkage convention is the common Workplace OS,
/ SunSoft, and embedded processor conventions based on GOT (Global Offset Table). Emulation capabilities
/ are available for 80486 ring 3, DOS 6.3, Windows 3.11, and WIN32s. OS/2 for PowerPC supports the FAT
/ (DOS), HPFS, and ISO 9660 file systems.

F.4 Operating System Configuration

Workplace OS is offered in a single scalable configuration. This configuration may be used as a client or developer workstation. It can also be used as a server through the addition of products such as the IBM LAN Server for Workplace OS.

F.5 Hardware Configuration Requirements

This section defines the minimum and recommended hardware configurations. In many cases performance improvements can be realized by configuring systems with larger or faster components. Because these configuration adjustments are application load dependent, no attempt has been made to recommend operationally tuned configurations.

The subsections below describe the configuration recommendations, which are summarized in Table 32.

F.5.1 Processor Subsystem

A PowerPC Reference Platform-compliant system must have a PowerPC compliant processor. Workplace
/ OS will support PowerPC 601, 603, 604 or compatible PowerPC processors running at supported frequencies. Performance will be better with higher frequencies and with PowerPC processors having performance-enhancing features such as larger internal cache.

F.5.2 Memory Subsystem

System Memory for a Workplace OS client workstation should be at least 8 MB (16 MB is recommended). For a developer workstation, System Memory should be 16 MB. Servers may require additional memory. Performance improvements can be realized with additional System Memory.

System ROM will not vary between configurations. The minimum size is determined by what a vendor needs to boot the machine into the state expected by the operating system as defined in Section 5.0, "Boot Process and Firmware."

Workplace OS does not require a cache external to the processor. The cache is recommended for developer and server configurations.

Workplace OS uses the Non-volatile Memory for configuration and global environment storage. This information is shared between multiple operating systems (or multiple versions of the same operating system) on the platform. Non-volatile Memory requirements are the same for all workstation configurations.

F.5.3 Storage Subsystems

The minimum hardfile size for a Workplace OS client workstation is 120 MB. A developer workstation requires at least 240 MB. The minimum size for a server depends upon its function, but should not be smaller than 320 MB.

/ A 1.44-MB 3.5-inch floppy drive is required for all workstations to allow information interchange. This
/ requirement can be met by having external or network access to such a floppy device. Workplace OS also supports a 2.88-MB 3.5-inch floppy drive.

A CD-ROM is required for all Workplace OS machines. The Workplace OS distribution media will be CD-ROM. An acceptable alternative to internal CD-ROM support is an external CD-ROM, or network access to a CD-ROM. All workstations benefit from high-access-speed CD-ROMs.

F.5.4 Human Interface Subsystem

The video system must be capable of showing 640x480x8. Higher resolutions and additional color depth are
/ recommended for clients and developers. When emulating DOS applications which require planar graphics
/ (e.g. games and Prodigy*), all configurations must have a VGA-compatible video system.

A keyboard and pointing device are required for all workstations.

/ A business audio device is required as part of the standard hardware configuration described in Section 2.0,
/ "Hardware Configuration."

F.5.5 Connectivity Subsystem

/ Workplace OS does not require a serial port, but a serial port is required as part of the standard hardware
/ configuration described in Section 2.0, "Hardware Configuration." Workplace OS supports EIA/TIA-232-E-compliant serial ports. Any configuration may contain one or more serial ports.

Workplace OS does not require a parallel port. Workplace OS supports 8-bit bidirectional Compatibility
/ Mode-compliant parallel ports. Initial versions of Workplace OS will support an ECP parallel port, but may
/ not exploit its performance advantages.

/ No network connection is required. Workplace OS does not provide network support, but this support may
/ be provided through add-on products.

F.5.6 Expansion Bus Interfaces

/ Workplace OS requires one and only one PCI bus.

Workplace OS supports PCMCIA, including socket services. A device driver must be included for non-standard devices.

Workplace OS supports multiple instances of SCSI interfaces.

/ Workplace OS has very complete support for IDE.

The ISA bus is optional.

/ F.5.7 Security and System Management

/ Workplace OS has no special requirements in this area.

F.6 Hardware Configuration Recommendations

The Workplace OS system which supports PowerPC Reference Platform-compliant systems will support additional equipment. In some cases, drivers will have to be supplied by either the system vendor or the device vendor.

Audio upgrades with better sound and MIDI support would be useful additions to facilitate HUMAN-CENTERED applications.

Alternative graphics adaptors may be supplied on PowerPC Reference Platform-compliant machines.
/ Workplace OS initially supports S3, Weitek, and Western Digital adaptors. Vendors would have to provide drivers for other adaptors.

/ Workplace OS requires VGA capabilities to emulate DOS programs which use VGA planar modes.

/ Tape drives are useful as backup devices and archive mechanisms. Workplace OS supports tape devices in
/ the same manner as it supports other peripheral devices, and will contain sample device drivers.

F.7 Boot Time Abstraction Requirements

Workplace OS uses boot devices defined by the firmware. Only those devices (e.g. video, keyboard, disk) known to the firmware participate in the boot process.

/ Hardware abstraction is provided in Workplace OS by Open Firmware or by a customized microkernel. A
/ shrink-wrapped Workplace OS can be used with a custom microkernel, but this combination will decrease
/ system's the ability to fully boot from CD-ROM.

F.8 Hardware Abstraction Layer

Workplace OS has been designed to be portable to multiple hardware platforms. The portable nature of Workplace OS is a result of the use of a microkernel as the foundation of the operating system.

The microkernel is a small, message-passing nucleus of systems software running in the most privileged state of the computer. It supports the rest of the operating system as a set of applications called servers. The microkernel encapsulates the processor-specific information and other hardware dependencies. The other operating system components (i.e. the servers) communicate with the microkernel through a rich set of interfaces. The interfaces shield the servers from the processor architecture and system structure.

The microkernel supports a basic set of system services: virtual memory management, tasks and threads, interprocess communications, I/O support and interrupt management, and processor services. Higher-level operating system services, such as file system support, device driver support and application interfaces, are supported by out-of-kernel independent servers. This structure ensures that system integrity and security is not compromised by either misbehaved applications or device drivers. The system can scale to support diverse hardware and software configurations.

In the Workplace OS architecture, traditional application programming interfaces are supported by servers called personalities. Workplace OS for PowerPC Reference Platform-compliant systems provides the OS/2 personality and DOS/Windows** 3.1 support in the first release.

Figure 61 shows the Workplace OS structure. The underlying hardware is managed by the microkernel. Device drivers, the file system, and the OS/2 personality are user-level processes (not privileged). Applications are written to the interfaces exported by the OS/2 personality. This structure allows Workplace OS to easily support multiple hardware variants while ensuring portability of the operating system and user applications.

F.9 Device Driver Model

There will be a separate Device Driver Development Kit (DDK) available from IBM. It contains all the documentation and sample code needed to develop device drivers.

In addition, DOS device drivers may be used to access devices from a virtual DOS session.

F.10 Multiprocessing Model

The IBM microkernel, the Workplace personalities, and other Workplace OS components are designed to
/ support SMP. Workplace OS has set no specific limit on the number of processors in an SMP.

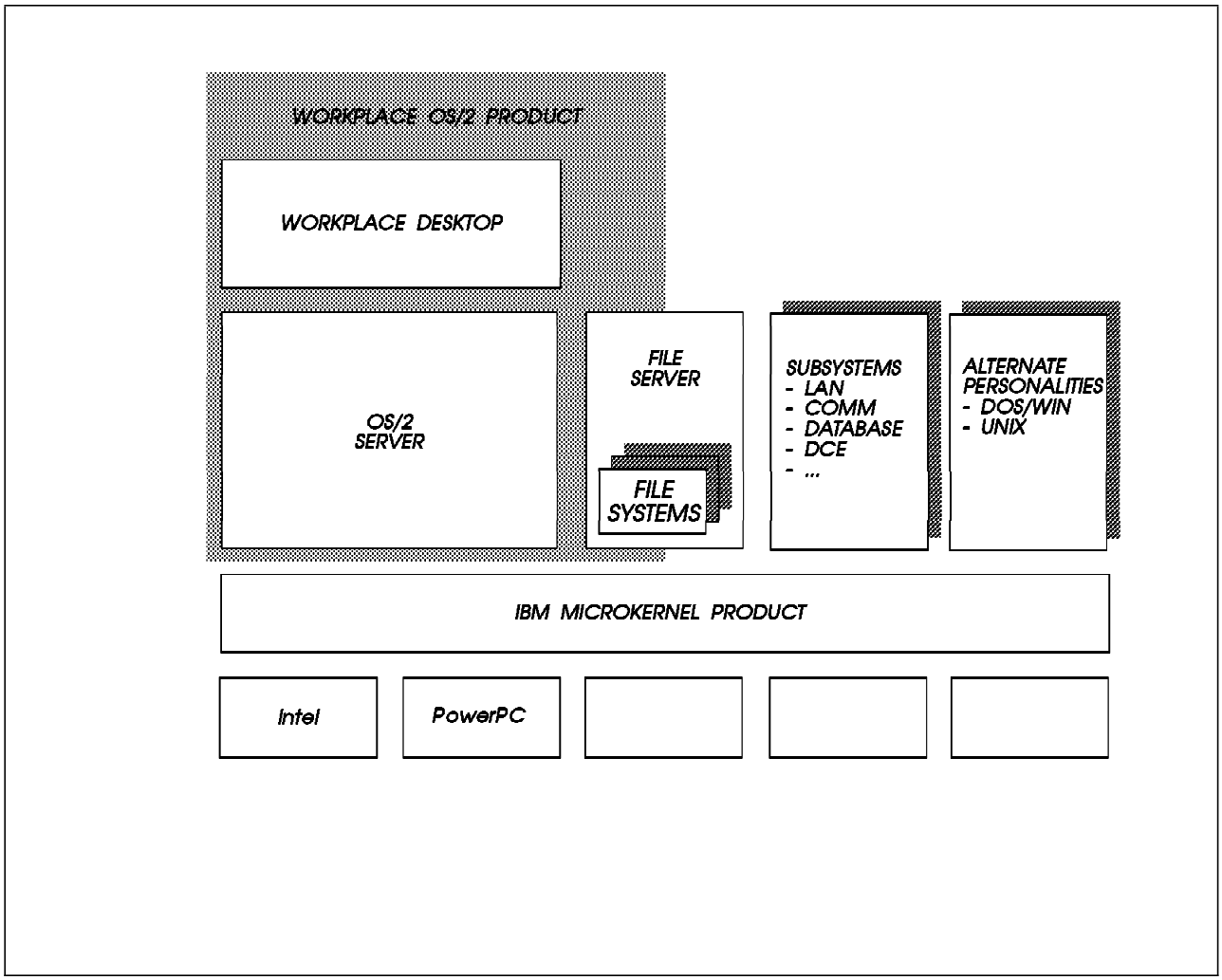


Figure 61. Workplace OS Structure

F.11 Application Model

Applications of Workplace OS/2 are written at several levels:

a) OS/2 32-bit PowerPC applications

b) DOS and Windows 16-bit Intel emulation applications

Thirty-two-bit applications written for OS/2 2.1 are source-compatible with the Workplace OS/2 personality. Such applications can make full use of the machine graphics capabilities using Presentation Manager.

Existing Intel x86-architecture DOS- and Windows-based applications can be run in DOS mode using an instruction set emulator (ISE). Workplace OS also provides a full emulation of both DOS and Windows.

Shared services and subsystems such as communication, database, and file servers can be written as Personality Neutral Services. Such subsystems can directly access the capabilities of the IBM microkernel and are available to applications as well as other subsystems.

F.12 Configuration Summary Table

Table 32 gives the minimum configuration information for three Workplace OS system configurations. This table specifies levels of hardware necessary to run these configurations. In many cases upgrade options are possible. Some of those options and configuration alternatives were described in Section F.5, "Hardware Configuration Requirements." As additional data for other personalities becomes available this table will be updated with similar data for each personality.

Note: The information in this table does not include the impact of DOS emulation software.

Table 32 (Page 1 of 2). Hardware Requirements for Workplace OS System Configurations

	System Component	Client	Developer	Server
/	Processor	PowerPC 601, 603, 604, or compatible processor		
	System Memory	8 MB	16 MB	> = 16 MB
	System ROM	Standard	Standard	Standard
	Non-volatile RAM	Standard 4 KB	Standard 4 KB	Standard 4 KB
	L2 Cache Memory	Optional	Optional	Optional
/	Hardfile	120 MB	240 MB	> = 320 MB
/	Floppy (1.44 MB)*	Required	Required	Required
/	CD-ROM*	Required	Required	Required
/	Keyboard	Required	Required	Required
/	Pointing Device	Required	Required	Required
/	Audio	Standard	Standard	Standard
	Graphics	640x480	640x480	640x480
	Real-Time Clock	Standard	Standard	Standard
/	Serial Ports	Standard	Standard	Standard
	EIA/TIA-232-E	Workplace OS supports this serial interface		
	EIA-422-A	Workplace OS does not support this serial interface		
/	Parallel Ports	Optional	Optional	Optional
	Compatibility Mode	Workplace OS supports this parallel interface		
/	Extended Capabilities Port	Workplace OS supports this parallel interface		
/	Network Adaptors	Optional	Optional	Required
/	SCSI	Optional	Optional	Optional
	IDE	Optional	Optional	Optional
	PCI Bus	1 Required	1 Required	1 Required
	ISA Bus	Optional	Optional	Optional
	PCMCIA	Optional	Optional	Optional

Table 32 (Page 2 of 2). Hardware Requirements for Workplace OS System Configurations

System Component	Client	Developer	Server
Tape Drive	Optional. A vendor-supplied device driver is required.		

Legend:

Entry	Definition
Standard	The hardware configuration in Section 2.0, “Hardware Configuration,” requires this component.
Required	This component must be present in systems on which Workplace OS will run.
Optional	This component is supported by Workplace OS, but is not required.
*****	Capability can be provided via a network.

Appendix G. Solaris

/ This appendix describes the Solaris** operating environment which runs on PowerPC Reference Platform-
/ compliant machines.

/ G.1 Operating System Scope

/ The Solaris operating environment is SunSoft's next-generation distributed computing solution. A fully
/ compliant implementation of System V Release 4 UNIX (SVR4), Solaris provides access to the most pow-
/ erful and advanced solutions available, including an open, fully networked environment. Solaris also unites
/ the world's largest installed base of CISC and RISC hardware -- SPARC, x86, and PowerPC.

/ The SunOS** 5.x operating system is the foundation of Solaris. The innovative technology it contains --
/ including the industry's leading implementation of a multithreading operating system -- brings a new level of
/ performance to Solaris users. SunOS 5.x builds on SVR4 and extends it by introducing new features
/ including symmetric multiprocessing with a multithreaded kernel, real-time functionality, advanced interna-
/ tionalization, and increased security features.

/ ONC+**, Solaris' core networking technology, is one of the world's most widely used heterogeneous net-
/ working solutions. The ONC+ family of protocols and distributed services, including NFS**, is supported
/ by over 300 companies and has an installed base of over 3 million users.

/ Solaris features OpenWindows**, SunSoft's X11, network-based windows system. With its windows, pull-
/ down menus, buttons, and drag-and-drop operations, OpenWindows provides a consistent, easy-to-use envi-
/ ronment for all types of users. The DeskSet** desktop productivity tools feature integrated, distributed tools
/ from basic time management applications through sophisticated workgroup communications tools. And
/ SunSoft's ToolTalk** interapplication communication solution facilitates information exchange between
/ applications on a single machine, or on multiple machines on a network.

/ The target market for Solaris is the Enterprise 1000. Large companies looking to rightsize their operations
/ will find Solaris an ideal solution for supporting heterogeneous hardware environments ranging from note-
/ books to high-end MP servers using x86, SPARC, and/or PowerPC architectures.

/ G.2 Operating System Version

/ This Appendix describes the first release of Solaris that will support the PowerPC architecture. This version
/ will provide users the same functionality that will be available to Solaris users on the SPARC and x86 archi-
/ tectures. The version number for this release of Solaris has not yet been announced.

/ G.3 Operating System Environment

/ Solaris runs on this PowerPC platform in Little-Endian mode. Solaris supports the ELF object module
/ format as defined in the SVR4 PowerPC ABI. Emulation programs which are not part of the operating
/ system are expected to be available to run applications from other environments, such as Windows, DOS, or
/ Macintosh. Solaris can import the non-native file systems High Sierra (i.e. ISO 9660) FAT (DOS), and S5
/ (UNIX System V).

G.4 Operating System Configuration

Solaris is offered in the following three configurations:

Desktop — This configuration is targeted for end-user and developer desktops, for use as clients on a network or as stand-alone workstations. It incorporates the full array of Solaris functionality, and comes with a one- to two-user license.

Workgroup Server — This configuration is targeted at departmental servers, for use as print, file, database, or application servers on a small network of up to 100 desktops. The license allows unlimited use but is restricted to platforms with a maximum of (currently) 2 CPUs.

Enterprise Server — This configuration supports high-end multiprocessor servers. There is no limit to the number of processors in the server, but the configuration is targeted primarily at MPs with no more than 20 processors. It supplements the basic Solaris functionality with additional components for disk management (RAID), networked backup, advanced network system administration, etc.

G.5 Hardware Configuration Requirements

This section defines the minimum and recommended hardware configurations. In many cases, performance improvements can be realized by configuring systems with larger or faster components. Because these configuration adjustments are application load dependent, no attempt has been made to recommend operationally tuned configurations.

The subsections below describe the configuration recommendations, which are summarized in Table 34.

G.5.1 Processor Subsystem

A PowerPC Reference Platform-compliant system must have a PowerPC compliant processor. Solaris will support PowerPC 601, 603, 604, or compatible PowerPC processors running at supported frequencies. Performance will be better with higher frequencies and with PowerPC processors having performance-enhancing features such as a larger internal cache.

G.5.2 Memory Subsystem

System Memory for a Solaris desktop must be at least 16 MB. Servers may require additional memory. Performance improvements can be realized with additional system memory.

System ROM will not vary between configurations. The minimum size is determined by what a vendor needs to boot the machine into the state expected by the operating system as defined in Section 5.0, “Boot Process and Firmware.”

A cache external to the processor is not required by Solaris, but is recommended for developer and server configurations.

Solaris uses the Non-volatile Memory for configuration and global environment storage. This information is shared between multiple operating systems or versions on the system. Non-volatile Memory requirements are the same for all configurations.

G.5.3 Storage Subsystems

There is no minimum hardfile requirement for Solaris, as diskless operation is fully supported. Alternative configurations provide dataless and cache-only support, requiring small local hardfiles (60 MB). For full installations, an end-user client requires 200 MB of hardfile space, and a developer workstation requires 320 MB of hardfile space. The minimum size for a departmental server is 320 MB. A server may require more space, depending on its function. A file server, for example, normally requires much more hardfile space, in the range of 2 to 4 GB.

A single 1.44-MB 3.5-inch floppy drive is required for all desktops. Solaris also supports a 2.88-MB 2.5-inch floppy drive.

A CD-ROM is required for all Solaris machines. The Solaris distribution media will be CD-ROM only. An acceptable alternative to internal CD-ROM support is an external CD-ROM, or network access to a CD-ROM coupled with booting of desktops over the network. All desktops benefit from high-access-speed CD-ROMs.

G.5.4 Human Interface Subsystem

All configurations should have an alphanumeric input and display device. If a server does not also function as a desktop, it may use a simple device such as a terminal connected to a serial port.

All desktops must have a graphics system capable of showing 640x480x8, although a recommended minimum is 1024x768x8. Higher resolutions and additional color depth are recommended for high-end graphics applications.

A keyboard and pointing device are required for all desktops.

A business audio device is required as part of the standard hardware configuration described in Section 2.0, “Hardware Configuration,” but is not required by Solaris for desktops. A higher-function audio device may be substituted.

G.5.5 Connectivity Subsystem

Solaris does not require a serial port, but a serial port is required as part of the standard hardware configuration described in Section 2.0, “Hardware Configuration.” Solaris supports EIA/TIA-232-E-compliant serial ports. Any configuration may contain more than one serial port.

Solaris does not require a parallel port. Solaris supports 8-bit bidirectional Compatibility Mode-compliant parallel ports.

No network connection is required, but if present, it is supported by all configurations. Network connectivity is strongly recommended, and is anticipated in the target markets for Solaris.

G.5.6 Expansion Bus Interfaces

Solaris supports the PCI, PCMCIA, and ISA buses. None of these are required, although a PCI bus is strongly recommended.

Solaris supports multiple instances of SCSI interfaces.

Solaris supports IDE access to disks.

G.6 Hardware Configuration Recommendations

Solaris will support additional equipment on PowerPC Reference Platform-compliant expansion buses. In some cases, drivers will have to be supplied by either the system vendor or the device vendor; in others, SunSoft will supply the drivers.

Alternative graphics adaptors may be supplied on PowerPC Reference Platform-compliant machines. Solaris initially supports S3 and Weitek adaptors, although this list will be expanded by SunSoft. Vendors are also encouraged to supply device support for other adaptors.

G.7 Boot Time Abstraction Requirements

Solaris uses boot devices defined by the firmware. Only those devices (video, keyboard, disk) known to the firmware participate in the boot process.

G.8 Hardware Abstraction Layer

Solaris has been ported to both Big-Endian (SPARC) and Little-Endian (Intel x86) processors. The same source code and hardware abstractions used for these processor architectures serves as the base for the PowerPC implementation of Solaris. Within each processor architecture, including PowerPC, Solaris supports multiple platforms.

Loadable modules provide dynamic tailorability for the Solaris kernel. Device drivers, bus interface modules (bus nexus drivers), file system implementations, scheduling algorithms, and STREAMS handling are all contained in Platform-Independent Modules (PIMs), which provide common support for all platforms on a processor architecture that implements a particular feature. For example, a loadable module might support a particular device or controller, a particular bus (e.g. PCI), a particular file system format (e.g. the ISO 9660 format for CD-ROMs), a particular scheduling class (e.g. real time), or a particular networking protocol. Such loadable modules are usually generated from source code that is common across processor architectures, though they might actually be used on only one processor architecture or one platform.

Platform-Specific Modules (PSMs), on the other hand, support functions whose implementation differs from one platform to another. The Kernel Binary Interface (KBI) spells out the interface to which independent hardware providers code platform support modules. The KBI is an extension and formalization of the technology that has been successfully employed for multiprocessor platform support on both SPARC and x86. Typically, supporting a new multiprocessor platform based on an already-supported processor takes less than a month of programmer effort.

A generic distribution of Solaris from SunSoft supports one or more base system configurations. It contains a kernel, a set of device and bus nexus drivers, a complete UNIX System V Release 4 (SVID-compliant) environment, Solaris Deskset tools, system administration software, and subroutine libraries.

The Solaris Device Driver Interface (DDI) provides a well-documented and stable base for independent device driver development. The DDI consists of a common base interface with minor extensions for each of the various processor architectures. Most drivers written for devices that work under Solaris on x86 can be recompiled without source change and run on PowerPC systems that have hardware (bus) support for the same devices. The DDI is supported by a Driver Development Kit (DDK), which consists of descriptions and technical documentation of the interfaces as well as sample drivers.

The generic Solaris distribution will support specified processors in the PowerPC family (support for the 601, 603, and 604 is planned for the initial release), so virtual memory management and TLB and internal cache

/ support are in PIMs provided by SunSoft; coherency of external system memory caches in PowerPC Refer-
/ ence Platform systems is maintained by hardware. Thus, although the KBI does support abstraction of
/ these functions, platform suppliers will usually need to supply a PSM that deals only with differences
/ between their platforms and those supported by the generic distribution in the following areas:

/ a) Interrupt controllers
/ b) Clock and timer functionality
/ c) Power management
/ d) (For multiprocessors only) initialization and inter-processor interrupts

/ Table 33 shows the Solaris components that implement each of the PowerPC Reference Platform
/ abstractions called for in Section 4.0, “Machine Abstractions.”

/ Table 33 (Page 1 of 2). Solaris Abstraction Components

/ **Required Abstraction**	**Solaris Implementation**			
/ /	**Open Firmware**	**Solaris KBI PIM**	**Solaris KBI PSM**	**Solaris DDI**
/ Boot-Time	X			
/ Run-Time Platform Data	X			
/ System Information	X	X		
/ System Memory	X	X		
/ I/O Device Information	X			X
/ Processor Initialization	X		X	
/ I/O Buffer Flushing				X
/ Virtual Memory		X		
/ TLB Flush		X		
/ TLB Reload		X		
/ Cache Management	X	X		
/ Interrupt Handling			X	X
/ DMA			X	X
/ Calendar & Timer Services			X	
/ I/O Addresses	X			X
/ Power Management	X		X	
/ Hardware Fault		X	X	X
/ Bus Nexus Drivers			X	X
/ PCI System Bus			X	
/ ISA System Bus			X	
/ SCSI Framework		X		
/ SCSI Host Bus Adapter				X
/ Device Drivers				X
/ Graphics				X
/ Keyboard				X

Table 33 (Page 2 of 2). Solaris Abstraction Components				
Required Abstraction	**Solaris Implementation**			
	Open Firmware	**Solaris KBI PIM**	**Solaris KBI PSM**	**Solaris DDI**
Super I/O				X
Others				X

G.9 Device Driver Model

A PowerPC Device Driver Development Kit (DDK) available from SunSoft will contain all the documentation and sample code needed to develop device drivers.

G.10 Multiprocessing Model

Solaris is designed to support symmetric multiprocessing. Solaris has no fixed limit to the number of processors, but it is tuned for up to 20. Platform-specific modules may need to be written to support a particular multiprocessing platform. Specifications for writing such modules will be available from SunSoft.

G.11 Application Model

Solaris supports native applications conforming to the System V Interface Definition (SVID), the generic System V application binary interface (gABI), and the PowerPC processor supplement to the System V ABI (PowerPC psABI).

At a source level, the APIs provided by Solaris are identical across all architectures supported by Solaris (SPARC, x86, PowerPC). Applications written to these APIs are independent of the underlying hardware, and require only recompilation to generate binaries that will run on Solaris across all supported architectures.

At a binary level, applications built to run on Solaris for PowerPC systems will run unchanged across all PowerPC platforms supported by Solaris.

Solaris will provide emulation software for running applications native to other operating environments, such as existing DOS and Windows applications built for the x86 architecture.

G.12 Configuration Summary Table

Table 34 gives the minimum configuration information for three Solaris system configurations. This table specifies levels of hardware necessary to run these configurations. In many cases upgrade options are possible. Some of those options and configuration alternatives were described in Section G.5, “Hardware Configuration Requirements.”

Table 34. Hardware Requirements for Operating System Configurations

System Component	Client	Developer	Server
Processor	PowerPC 601, 603, 604, or compatible processor		
System Memory	16 MB	16 MB	16 MB
System ROM	Standard	Standard	Standard
Non-volatile RAM	Standard 4 KB	Standard 4 KB	Standard 4 KB
L2 Cache Memory	Optional	Optional	Optional
Hardfile*	200 MB	320 MB	>=320 MB
Floppy (1.44 MB)	Required	Required	Required
CD-ROM*	Required	Required	Required
Keyboard	Required	Required	Required
Pointing Device	Required	Required	Optional
Audio	Standard	Standard	Standard
Graphics	640x480	1024x768	Optional
Real-Time Clock	Standard	Standard	Standard
Serial Ports	Standard	Standard	Standard
EIA/TIA-232-E	Solaris supports this serial interface		
EIA-422-A	Solaris does not support this serial interface		
Parallel Ports	Optional	Optional	Optional
Compatibility Mode	Solaris supports this parallel interface		
Extended Capabilities Port	Solaris does not support this parallel interface		
Network Adaptors	Optional	Optional	Required
SCSI	Optional	Optional	Optional
IDE	Optional	Optional	Optional
PCI Bus	Optional	Optional	Optional
ISA Bus	Optional	Optional	Optional
PCMCIA	Optional	Optional	Optional
Tape Drive	Optional	Optional	Optional

Legend:

Entry	Definition
Standard	The hardware configuration in Section 2.0, “Hardware Configuration,” requires this component.
Required	This component must be present in systems on which Solaris will run.
Optional	This component is supported by Solaris but is not required.
*	Capability can be provided via a network.

Appendix H. Taligent

Appendix I. PowerPC Supplement to IEEE 1275

I.1 Overview

This appendix specifies the application of *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements* to computer systems that use the PowerPC Instruction Set Architecture, including instruction set-specific requirements and practices for debugging, client program interface and data formats. An implementation of PowerPC Open Firmware must implement the core requirements as defined in [1] and the PowerPC specific extensions described in this binding.

This appendix refers to the publications listed in in Section I.2.1, "References," by number. For example, [2] refers to *The PowerPC Architecture*.

While this appendix addresses the official PowerPC architecture [2], the name "PowerPC" implies compliance only to Book I. The descriptions that follow, and the relevant sections describing translation features for this binding, assume that the system's PowerPC processor(s) implement the entire set of Books I-III. Some PowerPC processors may implement different Book II-III features; such processors may need a variant of this binding describing the differences to the mapping functions, etc.

This appendix defines the binding to PowerPC processors that use 32-bit addressing. Since the minimum cell size of Open Firmware is 32 bits, only one cell is necessary to represent addresses of processor bus devices; hence, the value of "#address-cells" for /root must be 1.

Note: Sixty-four-bit processors can follow the specifications contained herein as long as all addresses relevant to Open Firmware are kept within the first 4 GB; i.e. the upper 32 bits of 64-bit addresses are assumed to contain zeros.

I.1.1 Endianess Support

The implementation of PowerPC Open Firmware must support both Big- and Little-Endian system implementations. This section describes features added to the core Open Firmware features to support Bi-Endian booting.

I.1.2 Bi-Endian Booting

The Configuration Variable **little-endian?** must be implemented. The basic concept of Bi-Endian support is to keep in the **little-endian?** variable a "cached" indication of the desired Endianess of client programs (e.g. operating systems or their loaders). This variable indicates the expected Endian mode of a boot target; **false** (0) indicates Big-Endian, **true** (-1) indicates Little-Endian. The default value of **little-endian?** is implementation dependent.

The client program must describe its Endian mode in the header section of its image as described in Section I.8.1.1, "ELF Note Section." When Open Firmware is started, Open Firmware must use the value of **little-endian?** to establish the Endian mode of the system. After Open Firmware locates and loads a client program, the correct Endian mode must be verified with the description in the header section of the client program image. If the cached value is correct, Open Firmware proceeds, with the system in its current Endian mode.

If, however, Open Firmware determines that the Endian mode of the client program is different than it had assumed, it must set **little-endian?** appropriately and reconfigure the system (hardware and firmware) for the new Endian mode, possibly by resetting the system as with **reset-all**.

Note: The Endian mode applies to the entire hardware system, including the processor(s). Open Firmware must perform whatever steps are required to enable the system to run in the specified mode.

Client programs can use *setprop* to alter the value of **little-endian?**; users can alter it via the **setenv** command from the user interface (if present). The alteration of **little-endian?** must not cause the system to be reconfigured until the system is re-booted.

Note: This mechanism introduces an extra configuration pass. However, this occurs only when switching the Endian mode from that which was last used. For most boots, Open Firmware will be appropriately configured so that no additional overhead will occur.

I.1.3 PowerPC Address Translation Model

This section describes the model that is used for coexistence of Open Firmware and client programs (i.e. operating systems) with respect to address translation.

The following overview of translation is provided so that the issues relevant to PowerPC Open Firmware can be discussed. Details that are not relevant to Open Firmware issues (e.g. protection) are not described in detail; *The PowerPC Architecture* [2], particularly Book III, should be consulted for the details. For the scope of this section, terms will be used as defined in [2], and the 32-bit processor model is assumed.

I.1.3.1 Translation Requirements

The default access mode of storage for loads and stores (i.e. with translation disabled -- referred to as Real-Mode) for PowerPC microprocessors assumes that caches are enabled (in copy-back mode). In order to perform access to I/O device registers, the access mode must be set to Cache-Inhibited, Guarded by establishing a translation with this mode and enabling translation. Thus, even though most of a client program and/or Open Firmware can run with translation disabled, it must be enabled when performing I/O.

I.1.3.2 HTAB Translation

An effective address (EA) of a PowerPC processor is 32 bits wide. Each EA is then translated into a 52-bit virtual address (VA) by prepending a 24-bit virtual segment ID (VSID) to the 28 LSbs of the effective address; the VSID is obtained by indexing into a set of 16 segment registers (SRs) using the 4 MSbs of the EA. Finally, the virtual address is translated into a real address (RA). This is done by mapping the virtual page-number (VPN) (bits 0-39 of the VA) into a real page number (RPN) and concatenating this RPN with the byte offset (bits 40-51 of the VA). The mapping of VPN to RPN involves using a hashing algorithm within the hashed page table (HTAB) to locate a page table entry (PTE) that matches the VPN and using that entry's RPN component. If a valid entry is not found, a data storage interrupt (DSI) or instruction storage interrupt (ISI) is signalled, depending upon the source of the access.

Note: The translation from EA to VA via segment registers means that the granularity of unique effective (Open Firmware virtual) addresses (i.e. for a particular VSID) is 256 MB. This defines the size of, and rules for, the Open Firmware virtual address space described later (see Section I.1.5, "Open Firmware's Virtual-Mode Rules").

This process is not performed for every translation. Processors will typically have a translation lookaside buffer (TLB) that caches the most recent translations, thus exploiting the natural spatial locality of programs to reduce the overhead of address translation. On most PowerPC processors, the TLB updates are performed in hardware. However, the architecture allows an implementation to use a software-assisted mechanism to perform the TLB updates. Such schemes must not affect the architected state of the processor unless the translation fails, i.e. the HTAB does not contain a valid PTE for the VA and a DSI/ISI is signalled.

Note: One unusual feature of this translation mechanism is that valid translations might not be found in the HTAB; the HTAB might be too small to contain all of the currently valid translations. This introduces a level of complexity in the use of address translation by Open Firmware, as discussed below.

I.1.3.3 Block Address Translation

To further reduce the translation overhead for contiguous regions of virtual and real address spaces (e.g. a frame buffer, or all of real memory), the block address translation (BAT) mechanism is also supported by the PowerPC architecture. The block address translation involves the use of BAT registers that contain a block effective page index (BEPI), a block length (BL) specifier and a block real page number (BPRN); the architecture defines four BAT registers for data (DBAT registers) and four BAT registers for instruction (IBAT registers) [4].

Block address translation is done by matching some number of upper bits of the EA (specified by the BL value) against each of the BAT registers. If a match is found, the corresponding BPRN bits replace the matched bits in the EA to produce the RA.

Block address translation takes precedence over HTAB translation; i.e. if a mapping for a storage location is present in both a BAT register and the HTAB, the block address translation will be used.

I.1.4 Open Firmware's Use of Memory

Open Firmware must use the memory resources within the space indicated by the **"my-base"** and **"my-size"** Configuration Variables defined for PowerPC Open Firmware. As described in Section I.8.1.1, "ELF Note Section," a mechanism is defined to enable Open Firmware to determine if its current configuration is consistent with the requirements of the client. If it is not, Open Firmware must set these Configuration Variables appropriately and restart.

A PowerPC Open Firmware binding must support two different addressing models, depending upon the setting of the **real-mode?** Configuration Variable. This variable indicates the Open Firmware addressing mode that a client program expects; **false** (0) indicates Virtual-Mode, **true** (-1) indicates Real-Mode. The default value of **real-mode?** is implementation dependent.

The management of **real-mode?** is analogous to **little-endian?**. Open Firmware determines its addressing mode using the value of **real-mode?**. If the current state of **real-mode?** (and hence, the current state of Open Firmware) is incorrect, it must set **real-mode?** appropriately and reset itself, possibly by executing **reset-all**.

In the following two sections, some conventions in Real-Mode and Virtual-Address translations are described. Subsequent sections describe the assumptions that Open Firmware makes about the state and control of the system in regard to Open Firmware's use of system resources for three Open Firmware interfaces (e.g. Device, User, and Client interfaces).

I.1.4.1 Real-Mode

When **real-mode?** is **true**, Open Firmware must configure its address translation to run in Real-Mode. In Real-Mode, the uses of address translations by Open Firmware and its clients are considered independent; they do not share any translations. All interfaces between the two must pass addresses with the RA of the data. Any data structure shared by Open Firmware and its clients that refer to *virt* addresses in [1], or this binding, must be RAs (i.e. hardware addresses).

Note: In particular, the address of the client interface handler that is passed to the client has to be an RA.

---

[4] The 601 has a single set of BAT registers that are shared by both instruction and data accesses.

The Configuration Variables **“my-base”** and **“my-size”** must indicate the physical memory base and size in Real-Mode. The address base described with **“my-base”** and **“my-size”** does not include address space for I/O devices.

I.1.4.2 Virtual-Mode

When **real-mode?** is **false**, Open Firmware must configure itself to run in Virtual-Mode. In Virtual-Mode, Open Firmware and its client will share a single virtual address space. This binding provides interfaces to allow Open Firmware and its client to ensure that this single virtual address model can be maintained.

The Configuration Variables **“my-base”** and **“my-size”** indicate the virtual address space base and size in Virtual-Mode.

I.1.4.3 Device Interface (Real-Mode)

While Open Firmware is performing system initialization and probing functions, it establishes and maintains its own translations. In particular, it maintains its own HTAB (and/or BAT registers) and handles any DSI/ISI interrupts itself.

Note: In Real-Mode, all translations will be *virt=real*; the primary reason for translation is to allow appropriate I/O accesses.

I.1.4.4 Device Interface (Virtual-Mode)

Open Firmware will establish its own translation environment, handling DSI/ISI interrupts as in the Real-Mode case. However, this environment will, in general, contain *virt≠real* translations. The virtual address space used by Open Firmware must be compatible with its client.

Note: Since these virtual addresses will be used by the client and/or user interfaces (e.g. for pointers to its code, device tree, etc.), their translations must be preserved until the target OS decides that it no longer requires the services of Open Firmware.

I.1.4.5 Client Interface (Real-Mode)

In Real-Mode, addresses of client data are real; the client must ensure that all data areas referred to across the client interface are valid real addresses. This may require moving data to meet any requirements for contiguous storage areas (e.g. for **read/write** calls). Translation must be disabled before the client interface call is made.

Open Firmware will typically have to maintain its translations in order to perform I/O. Since the client may be running with translation enabled (except for the client interface call), Open Firmware must save the state of all relevant translation resources (e.g. SDR1, BAT registers) and restore them before returning to the client. Likewise, it may take over the DSI/ISI interrupts for its own use (e.g. for doing “lazy” allocation of BAT registers); it must preserve the state of any interrupt vectors for its client.

Since the state of the address translation system is not predictable to any interrupts, the client must ensure that interrupts are disabled before calling the client interface handler.

Once a client program starts execution, physical memory must be managed by the client program. Since Open Firmware does not know which physical memory the client program is using, Open Firmware must request physical memory, if needed, from the the client program (using **alloc-real-mem**).

I.1.4.6 Client Interface (Virtual-Mode)

Client interface calls are essentially "subroutine" calls to Open Firmware. Hence, the client interface executes in the environment of its client, including any translations that the OS has established; e.g. addresses passed in to the client interface are assumed to be valid virtual addresses within the scope of the OS. Any DSI/ISI interrupts are either invalid addresses or caused by HTAB "spills." In either case, the OS has the responsibility for the handling of such exceptions.

Note: Addresses that the Open Firmware internals use will be those that were established by the device interface (or, by subsequent actions of the client or user interface). Thus, the client must preserve these Open Firmware translations if it takes over the virtual memory management function.

In addition to using existing translations, the client interface might require the establishment of new translations (e.g. due to **map-in** calls during **open** time), or the removal of old translations (e.g. during **map-out** calls during **close** time). Since this requires altering the client's translation resources (e.g. HTAB), which might require handling spill conditions, Open Firmware cannot know how to perform these updates to the HTAB.

Hence, there must be *callback* services provided by the client for use by Open Firmware for such actions; see Section I.8.6.2, "Virtual Address Translation Assist Callbacks."

I.1.4.7 User Interface (Real-Mode)

In Real-Mode, Open Firmware has control of the system. As with the client interface in Real-Mode, it must save the state of the translation resources (including interrupt vectors) upon entry and must restore them upon exit.

I.1.4.8 User Interface (Virtual-Mode)

When the user interface is invoked, Open Firmware is responsible for managing the machine. Therefore, it will take over control of any relevant interrupt vectors for its own handling. In particular, it will take over DSI/ISI handling in order to report errors to the user for bad addresses, protection violations, etc. However, as described above, one source of DSI/ISI may simply be HTAB spills. As with the case of **map-in** and **map-out** calls, the user interface cannot know how to handle such spill conditions itself, or even if this is, in fact, a spill versus a bad address.

Hence, there must be a *callback* service provided by the client for use by Open Firmware to resolve such translations; see Section I.8.6.2, "Virtual Address Translation Assist Callbacks."

I.1.5 Open Firmware's Virtual-Mode Rules

In order to let clients (i.e. target operating systems) know where Open Firmware lives in the address space, the following rules must be followed by a PowerPC Open Firmware implementation and by client programs that make use of its client and/or user interfaces.

Open Firmware:

- must maintain its *"translations" "mmu"*-node property (see Section I.5.4, ""mmu" Node Properties")
- must not use BAT registers during client or user interface operations

Client programs:

- must provide callbacks to assist Open Firmware in address translation

I.2 References and Terms

I.2.1 References

This standard shall be used in conjunction with the following publications. When the following standards are superseded by an approved revision, the revision shall apply.

[1] *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements*.

[2] *The PowerPC Architecture*, published by IBM (ISBN 1-55860-316-6).

[3] *System V Application Binary Interface*, published by UNIX System Laboratories. This document describes the generic architecture of the ELF (Executable and Linking Format) object file format.

[4] *MS-DOS Programmer's Reference*, published by Microsoft. This document describes the MS-DOS partition, directory and FAT formats used by the *disk-label* support package.

[5] *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format*, found in the March 1994 issue of Microsoft Systems Journal.

[6] *Bootstrap Protocol*, Internet RFC 951; see also RFC 1532.

[7] *ANSI/NISO/ISO 9660, Information processing -- Volume and file structure of CD-ROM for information interchange*, published by the International Organization for Standardization.

[8] *System V Application Binary Interface, PowerPC Processor Supplement*, SunSoft. This document defines the PowerPC specific ABI for System V and also gives details on the PowerPC ELF format.

[9] *ISO 6429, Information processing -- ISO 7-bit and 8-bit coded character sets -- Additional control functions for character-imaging devices*, published by the International Organization for Standardization.

[10] *ANSI/IEEE X3.215-1994, Programming Languages -- Forth.*

[11] *ISO 639, Code for the representation of names of languages*, published by the International Organization for Standardization.

I.2.2 Terms

This standard uses technical terms as they are defined in the documents cited in Section I.2.1, "References," plus the following terms:

core, core specification — Refers to *IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Practices and Requirements*

ELF — Executable and Linking Format. A binary object file format defined by [3][8] that is used to represent client programs in PowerPC Open Firmware.

FDISK — Refers to the boot record and partition table format used by MS-DOS, as defined in [4].

linkage area — An area within the stack that is reserved for saving certain registers across procedure calls in PowerPC run-time models. This area is reserved by the caller and is allocated above the current stack pointer (%r1).

Open Firmware — The firmware architecture defined by the core specification or, when used as an adjective, a software component compliant with the core specification.

PE — Portable Executable. A binary object file format defined by [5]; this format is used by Microsoft's NT operating system.

procedure descriptor	A data structure used by some PowerPC run-time models to represent a C "pointer to procedure." The first word of this structure contains the actual address of the procedure.
Real-Mode	The mode in which Open Firmware and its client are running with translation disabled; all addresses passed between the client and Open Firmware are real (i.e. hardware) addresses.
Table of Contents (TOC)	A data structure used by some PowerPC run-time models that is used for access to global variables and for inter-module linkage. When a TOC is used, %r2 contains its base address.
Virtual-Mode	The mode in which Open Firmware and its client share a single virtual address space, and address translation is enabled; all addresses passed between the client and Open Firmware are virtual (translated) addresses.

I.3 Data Formats and Representations

The cell size must be 32 bits. Number ranges for *n*, *u*, and other cell-sized items are consistent with 32-bit, two's-complement number representation.

The required alignment for items accessed with *a-addr* addresses must be four-byte aligned (i.e. a multiple of 4).

Each operation involving a *qaddr* address must be performed with a single 32-bit access to the addressed location; similarly, each *waddr* access must be performed with a single 16-bit access. This implies four-byte alignment for *qaddrs* and two-byte alignment for *waddrs*.

I.4 Packages

This section describes the PowerPC specific requirements of Open Firmware packages.

I.4.1 "Disk-Label" Support Package

This section describes the partition formats and client program formats that the *disk-label* support package for PowerPC Open Firmware must support; an implementation may support additional mechanisms, in an implementation-specific manner.

The process of loading and executing a client program occurs in two stages. The first stage determines what file to read into memory. This is done by locating a file from the boot device, usually by means of a name lookup within a directory contained within a disk "partition." The second stage examines the front portion (header) of the image for "well-known" program formats. When the format of the image has been determined, the loading is completed in a manner determined by that format.

The name of the file (and the partition in which it is contained) can be explicitly specified by the user via the **load** or **boot** command, or can be implicitly specified by the value of the **"boot-device"** property of the *"/options"* node. The filename is the *ARGUMENTS* portion of the final *COMPONENT* of the *PATH_NAME*, as described in Section 4.3.1 of [1].

The syntax for explicit filename specification is as follows:

```
[n] [,filename]
```

where *n* is the partition number to be used and *filename* is the name of a file within that partition. If *n* is omitted, the default partition (as determined by the partition format) is used. If *filename* is omitted, the default filename (i.e. the filename component of the **"boot-device"** path name) is used. Partition number 0 refers to the entire device. This definition is independent of the existence of partition information.

I.4.1.1 Partition Formats

PowerPC Open Firmware must support FDISK-type partitions and ISO 9660, FAT-12, and FAT-16 file systems, as shown in the following algorithm:

```
Algorithm for locating boot file
    read sector 0 (bootsector)
    if last 2 bytes of sector are 0AA55h (little-endian)
         if bsMedia == 0F8h  \ FDISK partition on hard drive
             if an explicit partition has been requested
                  select partition number n
             else
                  select bootable partition (80h in peBootable field)
             use directory of the selected partition to locate file
         else (non-partitioned)
             use FAT-12/FAT-16 directory to locate file
    else
    read sector 16.
    if a valid ISO-9660 directory is found
         locate the file, using the ISO-9660 directory.
    else
         FAIL, in an implementation-specific manner.
```

The boot device is selected as described in 7.4.3.2 of [1]. A filename can be explicitly given as the arguments field of the *device-specifier* (i.e. the field following the ":" of the last path component). Other formats may be recognized in an implementation-specific manner.

Even though the above algorithm is not dependent on a certain device type, the following formats are strongly recommended for greater portability.

Floppy Disk

It is strongly recommended that floppy disks (1.44/2.88 MB, MFM) be in FAT-12 format, as described in [4].

Hard Disk

It is strongly recommended that hard disks have an FDISK partition map, as described in [4].

Note: Since the boot program is contained in the boot sector for floppies and the FDISK partition map for hard disks, the *"disk-label"* package must use the value of the *bsMedia* byte (located at offset 15h) to determine whether a partition map is present. If the value is 0F8h, it indicates that a hard disk and a partition map should be present in the boot sector; any other value indicates a floppy disk.

CD-ROM

It is strongly recommended that CD-ROMs be formatted according to ANSI/NISO/ISO 9660, as described in [7].

I.4.1.2 Program Image Formats

Open Firmware must recognize client programs that are formatted as ELF [3][8] and PE [5]. PE format support is provided only for booting Windows NT; all other clients must use ELF. Other formats may be handled in an implementation-specific manner.

After locating the file, Open Firmware reads the image into memory at the location specified by the **load-base** Configuration Variable. Then, Open Firmware must perform the following algorithm to prepare the image for execution.

```
init-program.
     examine the header of the image.
     set restart? false
     if the image is in ELF format
        if the EI_DATA field does not match little-endian?
              set little-endian? appropriately.
              set restart? true
        locate the Note Section for the PowerPC Reference Platform
        if the Note Section's descriptor is not correct
              set Configuration Variables appropriately
              set restart? true
        if restart?
              restart the system, possibly by executing reset-all
        else
              move and/or relocate the ELF image.
     else
     if the file is in PE format
          if little-endian? is false
             set little-endian? to true.
             restart the system, possibly by executing reset-all
        else
            move and/or relocate the PE image.
     else
         FAIL, in an implementation-specific manner.
```

I.4.2 "obp-tftp" Support Package

The *"obp-tftp"* Support Package of an Open Firmware implementation (used to **load** from *"network"* devices) must use the BOOTP protocol, as described in [6].

I.5 Properties

This section describes the standard properties of Open Firmware for PowerPC Open Firmware implementations.

I.5.1 Root Node Properties

The following properties of the root node ("/") must be created by an Open Firmware implementation. Note that the root node typically corresponds to the common processor bus in a PowerPC system.

"#address-cells" Standard property, encoded as with **encode-int**, that specifies the number of cells required to represent physical addresses on the processor bus; the value of *"#address-cells"* for the processor bus must be 1.

"clock-frequency" Standard property, encoded as with **encode-int**, that represents the primary system bus speed (in hertz).

I.5.2 CPU Node Properties

For each CPU in the system, a CPU node must be defined as a child of "/" (the root). The following properties apply to each of these nodes.

"device_type" Open Firmware standard property. The value of this property for CPU nodes must be *"cpu"*.

"cpu-version" Standard property, encoded as with **encode-int**, that represents the processor type. This value is obtained by reading the Processor Version Register of the CPU.

"clock-frequency" Standard property, encoded as with **encode-int**, that represents the internal processor speed (in hertz) of this node.

"timebase-frequency" Standard property, encoded as with **encode-int**, that represents the rate (in hertz) at which the PowerPC TimeBase and Decrementer registers increment.

I.5.2.1 TLB Properties

Since the PowerPC architecture defines the MMU as being part of the processor, the properties defined by Section 3.6.5 of [1] and the following MMU-related properties must be present under "cpu" nodes.

"tlb-size" Standard property, encoded as with **encode-int**, that represents the total number of TLB entries.

"tlb-sets" Standard property, encoded as with **encode-int**, that represents the number of associativity sets of the TLB. A value of 1 indicates that the TLB is fully associative.

I.5.2.2 Internal (L1) Cache Properties

The PowerPC architecture defines a Harvard-style cache architecture; however, unified caches are an implementation option. All of the PowerPC cache instructions act upon a cache "block" (also referred to as a cache "line"). The internal (also referred to as "L1") caches of PowerPC processors are represented in the Open Firmware device tree by the following properties contained within *"cpu"* nodes.

"cache-unified" This property, if present, indicates that the internal cache has a unified organization. Absence of this property indicates that the internal caches are implemented as separate instruction and data caches.

"i-cache-size" Standard property, encoded as with **encode-int**, that represents the total size (in bytes) of the internal instruction cache.

"i-cache-sets" Standard property, encoded as with **encode-int**, that represents the number of associativity sets of the internal instruction cache. A value of 1 signifies that the instruction cache is fully associative.

"i-cache-block-size" Standard property, encoded as with **encode-int**, that represents the internal instruction cache's block size, in bytes.

"d-cache-size" Standard property, encoded as with **encode-int**, that represents the total size (in bytes) of the internal data cache.

"d-cache-sets" Standard property, encoded as with **encode-int**, that represents the number of associativity sets of the internal data cache. A value of 1 signifies that the data cache is fully associative.

"d-cache-block-size" Standard property, encoded as with **encode-int**, that represents the internal (L1) data cache's block size, in bytes.

“l2-cache”	Standard property, encoded as with **encode-int**, that represents another level of cache in the memory hierarchy. Absence of this property indicates that no further levels of cache are present. If present, its value is the *phandle* of the device node that represents the L2 cache.

I.5.3 External (L2, L3...) Cache Properties

Some systems might include external (L2) cache(s). As with the internal caches, they can be implemented as either Harvard-style or unified. Unlike the L1 properties that are contained within the *“cpu”* nodes, L2 caches are contained within other device tree nodes.

The following properties define the characteristics of such L2 caches. These properties must be contained as a child node of one of the CPU nodes; this is to allow path-name access to the node. All *“cpu”* nodes that share the same L2 cache (including the CPU node under which the L2 cache node is contained) must contain an *“l2-cache”* property whose value is the *phandle* of that L2 cache node.

Note: It is possible to extend this scheme to arbitrary levels of secondary, tertiary, etc. caches. The *“l2-cache”* property must be used in one level of the cache hierarchy to represent the next level. The device node for a subsequent level must appear as a child of one of the caches in the hierarchy to allow path-name traversal.

“device-type”	Open Firmware Standard property; the device-type of L2-cache nodes must be *“cache”*.
“cache-unified”	This property, if present, indicates that the L2 cache has a unified organization. Absence of this property indicates that the L2 caches are implemented as separate instruction and data caches.
“i-cache-size”	Standard property, encoded as with **encode-int**, that represents the total size (in bytes) of the L2 instruction cache.
“i-cache-sets”	Standard property, encoded as with **encode-int**, that represents number of associativity sets of the L2 instruction cache. A value of 1 signifies that the instruction cache is fully associative.
“d-cache-size”	Standard property, encoded as with **encode-int**, that represents the total size (in bytes) of the L2 data cache.
“d-cache-sets”	Standard property, encoded as with **encode-int**, that represents number of associativity sets of the L2 data cache. A value of 1 signifies that the data cache is fully associative.
“l2-cache”	Standard property, encoded as with **encode-int**, that represents another level of cache in the memory hierarchy. Absence of this property indicates that no further levels of cache are present. If present, its value is the *phandle* of the device node that represents the cache at the next level.

I.5.4 "mmu" Node Properties

To aid a client in “taking over” the translation mechanism and still interact with Open Firmware (via the client interface), the client needs to know what translations have been established by Open Firmware. A standard property for “mmu” nodes (i.e. nodes whose *“device_type”* = *“mmu”*) is defined by the PowerPC binding.

"translations" This property, consisting of sets of translations, defines the currently active translations that have been established by Open Firmware (e.g. using *map*). Each set has the following format:

```
( virt size phys mode )
```

Each value is encoded as with **encode-int**.

I.5.5 "/openprom" Node Property

Open Firmware must implement the "halt-address" property under the "/openprom" node. The property value describes the location of physical memory where the firmware saves the halt procedure image. The halt procedure is the self-contained firmware callback service for system power-off or reboot. The halt procedure is discussed further in Section I.8.7, "Halt Callback."

"halt-address" This property describes the physical memory location of the halt procedure. The value has the following format:

```
:prop-encoded-array: address, length
```

If the halt procedure is located within read-write memory, *length* is the number of bytes of memory, encoded as with **encode-int**, occupied by the halt procedure. If the halt procedure is located within read-only memory, *length* is 0.

Note: "Read-only memory" refers to memory that cannot be written in normal operation; ROM or Flash ROM is considered to be "read-only," while RAM, whether or not it is write-protected via an MMU, is considered to be "read-write."

The read-write memory, if any, that the halt procedure occupies must not be included within the "available" property of the "/memory" node.

I.6 Methods

The MMU method defined by Section 3.6.5 of [1] must be implemented by CPU nodes. The value of the **mode** parameter for the relevant methods (e.g. **map**) must be the value that is contained within PTEs that control Write-Through, Cache-Inhibit, Memory-Coherent, Guarded and the 2 protection bits; thus, its format is: WIMGxPP, where x is a reserved bit that must be 0. In order for I/O accesses to be properly performed in a PowerPC system, address ranges that are mapped by **map-in** must be marked as Cache-Inhibited, Guarded.

I.7 Client Interface Requirements

A PowerPC Open Firmware implementation must implement a client interface (as defined in Chapter 6 of [1]) according to the specifications contained within this section.

I.7.1 Calling Conventions

Table 35 (Page 1 of 2). Register Usage Conventions

Register(s)	Value	Notes
%msr	preserved by client interface	
%cr	preserved by client interface	1
%r1-%r2	preserved by client interface	

Table 35 (Page 2 of 2). Register Usage Conventions

Register(s)	Value	Notes
%r3	argument array address on client interface entry	2
	result value (**true** or **false**) on client interface return	2
%r13-%r31	preserved by client interface	
%sprg0-%sprg3	preserved by client interface	
%fpscr	preserved by client interface	
%f0-%f31	preserved by client interface	
%lr, %ctr, %xer	undefined	
Other SPRs	preserved by client interface	3

Notes:

1. Only the non-volatile fields (cr2-cr4) need to be preserved.

2. As defined by Section 6.3.1 of [1]

3 Other special purpose registers

To invoke a client interface service, a client program constructs a client interface argument array as specified in the core Open Firmware document, places its address in %r3 and transfers to the client interface handler, with the return address in %lr. (A typical way of accomplishing this is to copy the client interface handler's address into %ctr and execute a *bctrl*.)

The client interface handler must perform the service specified by the contents of the argument array that begins at the address in %r3, place the return value (indicating success or failure of the attempt to invoke the client interface service) back into %r3, and return to the client program. This is typically done by a Branch to Link Register (*blr*).

The client interface handler must preserve the contents of the Stack Pointer (%r1), TOC Pointer (%r2), Condition Register (%cr), all non-volatile registers (%r13-%r31), and all special purpose registers except %lr, %ctr and %xer.

The preservation of %r2 allows TOC-based client programs to function correctly. Open Firmware must not depend upon whether its client is TOC-based or not. If the client interface handler itself is TOC-based, it must provide for the appropriate initialization of its %r2.

I.8 Client Program Requirements

I.8.1 Client Program Format

The data format of a client program compliant with this specification must be either ELF (Executable and Linkage Format) as defined by [3][8], and extended by Section I.8.1.1, "ELF Note Section," or PE (Portable Executable) as defined by [7]. The standard ELF format contains explicit indication as to the program's execution modes (e.g. 32- or 64-bit, Big- or Little-Endian); this version of the specification deals only with the 32-bit version (i.e. ELFCLASS32).

Note: Other client program formats may be supported, in an implementation-specific manner, by an Open Firmware implementation.

A standard client program must be statically linked, requiring no relocation of the image. The program's entry point (e_entry) must contain the address of the first PowerPC instruction of the client program. If the client program uses a TOC, the client program must establish the appropriate value of %r2.

Note: The entry point is the address of the first instruction of the client program, not that of a procedure descriptor.

I.8.1.1 ELF Note Section

Part of the process of loading a client program involves verifying its environmental requirements (e.g. Endianess and address translation mode) against the current firmware configuration. The client's Endianess can be directly determined by examining the ELF EI-DATA value; ELFDATA2LSB (1) implies Little-Endian while ELFDATA2MSB (2) implies Big-Endian. However, the other client requirements (e.g. address translation mode) are defined by means of an ELF Note Section (SHT_NOTE). The following describes the format of the Note Section for a client program file.

As defined by [3], an ELF file can be "annotated" by means of Note Sections within the executable file. A Note Section contains a "header" followed by a (possibly null) "descriptor," as follows:

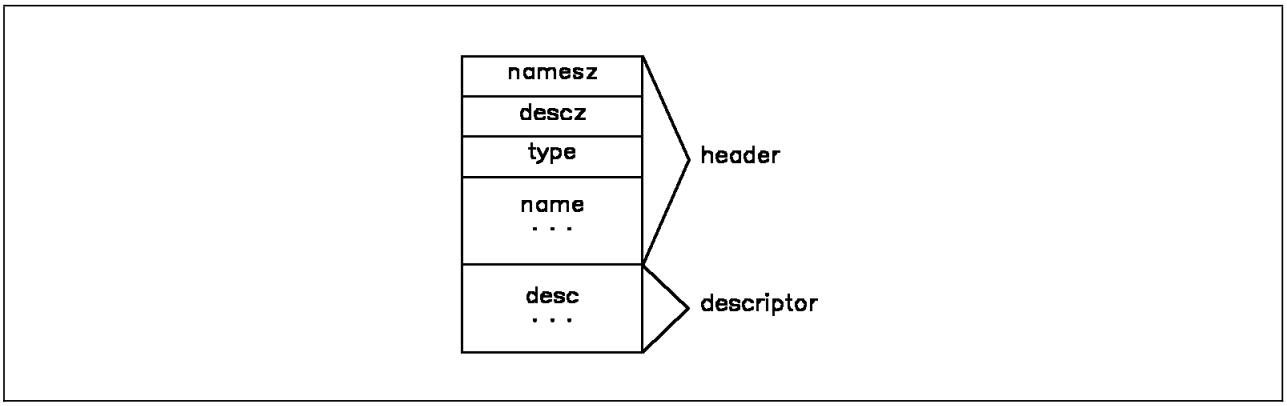


Figure 62. ELF Note Section

Note: The Endian format of the values corresponds to the Endianess specified by the EI-DATA field of the file.

The format of a Note Section header can be described by an Open Firmware *struct* as:

```
struct        \ Note Section header for PowerPC Reference Platform Open Firmware
/L  field         ns.namesz           \ length of ns.name, including NULL
/L  field         ns.descrsz
/L  field         ns.type
0   field         ns.name             \ NULL-terminated, /L padded
```

The *ns.name* field of the PowerPC Open Firmware Note Section must be *"PowerPC"*; the *ns.type* field must be 0x1275.

Following the Note Section header is a descriptor (*desc*); the length (in bytes) of the descriptor is specified by a word in the Note Section header (*descsz*). The interpretation of the descriptor depends upon the kind of Note Section in which it is contained. For Open Firmware, the format of the Note Section's descriptor can be described by an Open Firmware *struct*, as follows:

```
struct    \ Note Section descriptor for PowerPC Reference Platform Open Firmware
/L  field         ns.real-mode
/L  field         ns.my-base
/L  field         ns.my-size
```

I.8.2 Load Address

The load address is specified by the value of the **load-base** Configuration Variable. At least 4 MB of memory must be available at that address. It is strongly recommended that as much memory as is practical for the particular system be available there, thus allowing the loading of large client programs. Note that this address represents the area into which the client program file will be read by **load**; it does not correspond to the addresses at which the program will be executed.

I.8.3 Initial Program State

This section defines the "initial program state," the execution environment that exists when the first machine instruction of a client program of the format specified above begins execution. Many aspects of the initial program state are established by **init-program**, which sets the saved program state so that subsequent execution of **go** will begin execution of the client program with the specified environment.

I.8.3.1 Initial Register Values

Upon entry to the client program, the following registers must contain the following values:

Table 36 (Page 1 of 2). Initial Register Values

Register(s)	Value	Notes
%msr	EE=0, interrupts disabled	1
	PR=0, supervisor state	
	FP=1, floating-point enabled	
	ME=1, machine checks enabled	
	FE0, FE1=0, floating-point exceptions disabled	
	IP see Section I.8.5, "Interrupts"	
	IR, DR see Section I.1.4.1, "Real-Mode"	
	SF=0, 32-bit mode	
	ILE, LE see Section I.1.2, "Bi-Endian Booting"	2
%r1	See Section I.8.3.2, "Initial Stack"	
%r2	0	3
%r3	Reserved	4
%r4	Reserved	4
%r5	See Section I.8.3.3, "Client Interface Handler Address"	
%r6, %r7	See Section I.8.3.4, "Client Program Arguments"	

Table 36 (Page 2 of 2). Initial Register Values

Register(s)	Value	Notes
Others	0	

Notes:

1. Open Firmware will typically require the use of external interrupts for its user interface. However, when a client program is invoked, external interrupts must be disabled. If a client program causes the invocation of the user interface, external interrupts may be re-enabled.
2. The 601 processor uses a different mechanism for controlling the Endian mode of the processor. On the 601, the LE bit is contained in the HID0 register; this bit controls the Endian mode of both program and privileged states.
3. Open Firmware does not make any assumptions about whether a client program is TOC-based or not. It is the responsibility of the client program to set %r2 to its TOC, if necessary.
4. As defined in Section 5.6, “Residual Data.”

I.8.3.2 Initial Stack

Client programs must be invoked with a valid stack pointer (%r1) with at least 32 KB of memory available for stack growth. The stack pointer must be 16-byte aligned, reserving sufficient room for a linkage area (32 bytes above the address in %r1). If the system is executing in Real-Mode, the value in %r1 is a real (hardware) address; if in Virtual-Mode, the address in %r1 is a mapped virtual address.

I.8.3.3 Client Interface Handler Address

When client programs are invoked, %r5 must contain the address of the entry point of the client interface handler. If the system is executing in Real-Mode, the value in %r5 is a real (hardware) address; if in Virtual-Mode, the address in %r5 is a mapped virtual address.

Note: This address points to the first instruction of the client interface handler, not to a procedure descriptor.

I.8.3.4 Client Program Arguments

The calling program may pass to the client an array of bytes of arbitrary content; if this array is present, its address and length must be passed in registers %r6 and %r7, respectively. For programs booted directly by Open Firmware, the length of this array is zero. Secondary boot programs may use this argument array to pass information to the programs that they boot.

Note: The Open Firmware standard makes no provision for specifying such an array or its contents. Therefore, in the absence of implementation-dependent extensions, such an array will not be passed to a client program executed directly from an Open Firmware implementation. However, intermediate boot programs that simulate or propagate the Open Firmware client interface to the programs that they load can provide such an array for their clients.

Note: **boot** command line arguments, typically consisting of the name of a file to be loaded by a secondary boot program followed by flags selecting various secondary boot and operating system options, are provided to client programs via the “*bootargs*” and “*bootpath*” properties of the “*/chosen*” node.

I.8.4 Caching

The caches of the processor must be enabled when the client program is called. Memory areas allocated on behalf of the client program must be marked as cacheable. Accesses to I/O devices (especially to devices across bus bridges) must be made with the register access words (e.g. *rl@*) using virtual addresses returned by **map-in**.

I.8.5 Interrupts

Open Firmware requires that interrupts be "vectored" to its handlers when it is in control of the processor; this will occur when the user interface is running. Client interface calls are considered to execute in the environment of the client, and hence do not assume ownership of interrupts.

In order for Open Firmware to process interrupts in an efficient manner, an area of memory for the exclusive use by Open Firmware must be reserved by the client program at (real) memory locations 0x1E0...0x1FF.

Open Firmware must save and restore the first location of each interrupt that it wants to "take over"; i.e. whenever Open Firmware needs the use of an interrupt, it must save the current contents of the corresponding entry point and replace that location with a branch to its entry point. When Open Firmware returns control, it must restore the RAM location to its original contents.

I.8.6 Client Callbacks

This section defines the callback mechanism that allows Open Firmware to access services exported to it by the client program. As described in Section 6.3.2.6 (and the glossary entries for *callback* and *$callback*) in [1], the callback mechanism follows the same rules as those of client interface calls; i.e. an argument array is constructed by Open Firmware and the address of that array is passed (via %r3) to the client's callback routine. The address of the callback routine is supplied to Open Firmware by means of the *set-callback* client call.

If the system is running in Real-Mode, the address of the client callback routine must be a real (hardware) address; if it is running in Virtual-Mode, the client callback routine address must be a mapped virtual address.

I.8.6.1 Real-Mode Physical Memory Management Assist Callback

Once the control of physical memory is transferred to the client program, Open Firmware which is running in Real-Mode must use the callback service provided by the client program to allocate physical memory. Client programs that expect Open Firmware to operate in Real-Mode must implement the following physical memory management routine for Open Firmware:

```
alloc_real_mem
     IN: [address] min_addr, [address] max_addr, size, mode
     OUT: error, [address] real_addr
```

This routine allocates a contiguous physical memory of *size* bytes within the address range between *min_addr* and *max_addr*. The *mode* parameter contains the WIMGxPP bits as defined in Section I.6, "Methods." A non-zero error code must be returned if the mapping cannot be performed. If error code is zero (i.e. allocation has succeeded) the routine returns the base address of the physical memory allocated for Open Firmware.

I.8.6.2 Virtual Address Translation Assist Callbacks

As mentioned in Section I.1.5, “Open Firmware’s Virtual-Mode Rules,” when Open Firmware is running in Virtual-Mode, client programs that take over control of the system’s memory management must provide a set of callbacks that implement MMU functions. This section defines the input arguments and return values for these callbacks. The notation follows the style used in Chapter 6 of the Open Firmware specification [1].

```
map
     IN:  [address] phys, [address] virt, size, mode
     OUT: error
```

This routine creates system-specific translation information; this will typically include the addition of PTEs to the HTAB. If the mapping is successfully performed, a value of zero must be placed in the *ret1* cell of the argument array; a non-zero error code must be returned (in *ret1*) if the mapping cannot be performed.

```
unmap
     IN:  [address] virt, size
     OUT: none
```

The system removes any data structures (e.g. PTEs) for the virtual address range.

```
translate
     IN:  [address] virt
     OUT: error, [address] real.hi, [address] real.lo, mode
```

The system attempts to compute the real address (*real*) to which the virtual address (*virt*) is mapped. If the translation is successful, a PTE must be placed into the HTAB for this translation, the resulting real address must be returned in *real.hi,real.lo* and *ret1* must be set to **false** (0). If the translation is not successful, *ret1* must be set to a non-zero error code.

This call must be made when Open Firmware handles a DSI/ISI within the user interface. A successful result of the translate call indicates that Open Firmware can complete the interrupted access; a failure indicates that an access was made to an invalid address.

I.8.7 Halt Callback

The halt callback allows a client program to invoke limited firmware services for turning off or rebooting the machine. The halt procedure must not return control to the client program that invoked it. Consequently, the halt procedure is not required to preserve any machine state on behalf of its caller.

PowerPC Open Firmware must implement the “halt-address” property within the “/openprom” node to support halt callback. Detailed discussion about this property is presented in Section I.5.5, “"/openprom" Node Property.”

I.8.7.1 Halt Procedure Calling Conventions

When control is transferred to the halt procedure, the system must be in the Endian mode that was in effect when the firmware last had control of the system. The client program must establish the Real-Mode address translation for the firmware.

I.8.7.2 Halt Procedure Argument

The client program must set GPR3 to point to the halt argument. The halt argument is the address of a null-terminated text string. In the following discussion, this string is called the “halt string.”

If the halt string is “power-off,” the firmware must turn off the system power to the extent possible. If the firmware cannot turn off the system power, perhaps due to lack of hardware capability, the result is unde-

fined, and the firmware must not try to return control to the client program or reboot the system. The suggested behavior in this case is to enter a firmware-interactive mode, if available.

The firmware may recognize strings other than “power-off” in a system-dependent manner.

If the halt string is not a recognized command, the firmware must reboot the system. During the system reset and the firmware restart, the firmware must preserve the halt string. Then, the firmware must evaluate the string as if “auto-boot?” were true and “boot-command” were set to that string, without altering the values of the “auto-boot?” and “boot-command” Configuration Variables. If the firmware includes an Open Firmware user interface, the string can be any valid user interface command string. Otherwise, the firmware must interpret the string as shown in Table 37.

Table 37. Halt Arguments

String Value	Meaning
boot	Load and execute the default client program
<empty string>*	If the firmware has an interactive mode, enter that mode. Otherwise the result is undefined, except that the firmware must not perform the “boot” action.
<any other string>	Behavior is system-dependent
Note: *<empty string> means a valid address that points to a null character. An address value of 0 is not an empty string.	

I.9 User Interface Requirements

An implementation of PowerPC Open Firmware must conform to the core requirements as specified in [1] and the following PowerPC specific extensions.

I.9.1 Machine Register Access

The following user interface commands represent PowerPC registers within the saved program state. Executing the command returns the saved value of the corresponding register. The saved value may be set by preceding the command with **to**; the actual registers are restored to the saved values when **go** is executed.

I.9.1.1 Branch Unit Registers

%cr Access saved copy of Condition Register.

%ctr Access saved copy of Count Register.

%lr Access saved copy of Link Register.

%msr Access saved copy of Machine State Register.

%srr0 and %srr1 Access saved copy of Save/Restore Registers.

I.9.1.2 Fixed-Point Registers

%r0 through %r31 Access saved copies of Fixed-Point Registers.

%xer Access saved copy of XER Register.

%sprg0 through %sprg3 Access saved copies of SPRG registers.

I.9.1.3 Floating-Point Registers

Unlike the other registers, the floating-point unit registers are not normally saved, since they are not used by Open Firmware. The following access words, therefore, access the registers directly.

%f0 through %f31 Access Floating-Point Registers

%fpscr Access Floating-Point Status and Control Register

The following command displays the PowerPC CPU saved program state.

.registers

I.10 Configuration Variables

In addition to the standard Configuration Variables defined by the core Open Firmware document [1], the following Configuration Variables must be implemented for PowerPC Open Firmware:

"little-endian?" This Boolean variable controls the Endian mode of Open Firmware. If **true** (-1), the Endian mode is Big-Endian; if **false** (0), the Endian mode is Little-Endian. The default value is implementation dependent.

"real-mode?" This Boolean variable controls the address translation mode of Open Firmware. If **true** (-1), the addressing mode is Real-Mode; if **false** (0), the addressing mode is Virtual-Mode. The default value is implementation dependent.

"my-base" This integer variable defines the starting address to be used by Open Firmware. When Open Firmware is running in Real-Mode, **"my-base"** must be the physical memory address.

"my-size" This integer variable defines the size of the address space which is used by Open Firmware. When Open Firmware is running in Real-Mode, **"my-size"** must be the physical memory size.

"load-base" This integer variable defines the default load address for client programs when using the **load** method. The default value is implementation dependent.

I.11 Terminal Emulator Support Package

IEEE Std 1275-1994 defines the behavior of the Terminal Emulator Support Package (see Annex B of [1]). Annex B of [1] assumes that the Terminal Emulator Support Package implements certain escape sequences from the set defined by ANSI X3.64. The extension described here corresponds to ISO 6429-1983, as defined by [9]. An implementation of PowerPC Open Firmware is required to support additional graphic renditions (via SGR (Select Graphic Rendition) -- **ESC**[**#m**) beyond those specified in Annex B of [1].

In order for these extensions to be used, the FCode device driver for a display device (i.e. a device whose *"device_type"* property has the value *"display"*) must initialize the first 16 entries of its color table to appropriate values (see Section I.12.1.1, "Color Table Initialization"); note that this setup is standard for "VGA"-type devices. These values assume that the color is represented by the three LSbs of the color index and that the fourth LSb which corresponds to a value of 8 represents the intensity. The ISO 6429-1983 standard provides parameter values to independently control the color of foreground (30-37) and background (40-47). The intensity is set separately (1-2), and must be issued before the color control.

The expanded model of the Terminal Emulator is that there is a "current" background and foreground color (index), each of whose value is 0-15, corresponding to the first 16 entries of the color table. In positive

image mode, pixels corresponding to a font (logo) bit set (1) must be set to the foreground color; pixels corresponding to a font (logo) bit clear (0) must be set to the background color. When in negative image mode, the roles of foreground and background are reversed.

The default rendition must be positive image mode, background=15 and foreground=0, thus producing black characters on a bright white background.

The following table describes the effect of executing the SGR escape sequence with the specified parameter.

Table 38. SGR Parameters

Parameter	Interpretation
0	Default rendition
1	Bold (increased intensity)
2	Faint (decreased intensity)
7	Negative image
27	Positive image
30	Black foreground
31	Red foreground
32	Green foreground
33	Yellow foreground
34	Blue foreground
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black background
41	Red background
42	Green background
43	Yellow background
44	Blue background
45	Magenta background
46	Cyan background
47	White background

I.11.1 Display Device Low-Level Interfaces

PowerPC Open Firmware must implement the following method in the terminal emulation support package.

draw-logo-in-color (line# addr width height --) Draw (at line#) the logo stored at location *addr*

This method implements 8-bit-per-pixel color drawing. The FCode device driver for a display device must initialize the first 16 entries of its color table to the appropriate values as defined in Section I.12.1.1, “Color Table Initialization.”

In addition to the methods defined in Section 3.8.4.3 of [1], execution of **is-install** creates the **draw-logo-in-color** method in the current package that will execute the draw-logo-in-color deferred word.

When the *"fb8"* generic frame buffer package implements the display device low-level interface for a frame buffer, execution of **fb8-install** must install the **fb8-draw-logo-in-color** routine as the behavior of the draw-logo-in-color deferred word.

fb8-draw-logo-in-color (line# addr width height --) Implement the "fb8" draw-logo-in-color

I.12 Extensions for PowerPC Based Systems

This section describes the properties, methods and device subtrees that are applicable to devices required by the PowerPC Reference Platform system architecture. It is strongly recommended that other platforms follow these definitions for the corresponding devices.

I.12.1 Display Devices

This section defines additional behavior of display devices (e.g. *device_type = "display"*) for PowerPC Reference Platform Open Firmware implementations.

I.12.1.1 Color Table Initialization

The *PowerPC Reference Platform Specification* requires that display devices support a minimum of 256 colors. The core specification of Open Firmware defines a Terminal Emulation Support Package that, while defined for 8-bit pixels, does not include support for colors. Open Firmware implementations for the PowerPC Reference Platform must support additional Select Graphic Rendition parameters (see Section I.11, "Terminal Emulator Support Package") in order to allow client programs to display characters (and logo) using a 16-color model.

For this expanded Terminal Emulation support to work, Open Firmware device drivers for *"display"* devices must initialize the first 16 entries of their color table to values defined in Table 39. Note that the table values are defined in terms of percentage of full saturation color for each of the primary RGB colors.

Table 39 (Page 1 of 2). Color Table Values

Index	Red	Blue	Green	Color
0	0	0	0	Black
1	0	0	2/3	Blue
2	0	2/3	0	Green
3	0	2/3	2/3	Cyan
4	2/3	0	0	Red
5	2/3	0	2/3	Magenta
6	2/3	1/3	0	Brown
7	2/3	2/3	2/3	White
8	1/3	1/3	1/3	Gray
9	1/3	1/3	1	Light Blue
10	1/3	1	1/3	Light Green
11	1/3	1	1	Light Cyan

Table 39 (Page 2 of 2). Color Table Values				
Index	**Red**	**Blue**	**Green**	**Color**
12	1	1/3	1/3	Light Red
13	1	1/3	1	Light Magenta
14	1	1	1/3	Yellow
15	1	1	1	Bright White

I.12.1.2 "Display" Device Standard Properties

In addition to the standard properties defined by Open Firmware for display devices, the following properties must be present for PowerPC Reference Platform-compliant implementations.

"width" Standard property, encoded as with **encode-int**, that represents the visible width of the display in pixels.

"height" Standard property, encoded as with **encode-int**, that represents the visible height of the display in pixels.

"linebytes" Standard property, encoded as with **encode-int**, that represents the address offset between a pixel on one scan line and that same relative horizontal pixel position on the immediately following scan line.

"depth" Standard property, encoded as with **encode-int**, that represents the number of bits in each pixel.

I.12.2 Keyboard Devices

Open Firmware does not have a specific device class for keyboards; instead, keyboards are instances of the *"serial"* device class. However, certain features of keyboards (i.e. the ability to re-map keys, etc.) make it desirable to implement them as a separate device class.

In general, keyboard devices produce a hardware scan-code that is specific to the type of keyboard. These scan-codes are then mapped via software to produce the character code for the key, taking into account the state of "modifier" keys (e.g. Shift, Control) and the keyboard layout. The mapping of scan-codes to character value depends upon the keyboard layout. This is dependent upon the language that is being supported by the keyboard; e.g. the layout of keys for a French keyboard is different than that for an English keyboard.

For purposes of localization, it is necessary for the scan-code conversion to be controlled by the software, including Open Firmware (via the client interface). This section adds mechanisms to allow client programs to alter the scan-code conversion, based upon the keyboard (language) layout.

The language-specific layout is specified by means of a two-character abbreviation, as described by [11]. The following languages may be supported:

CS Czech

DA Danish

NL Dutch

EN English (default)

FI Finnish

FR French

| DE German

| HU Hungarian

| IT Italian

| NO Norwegian

| PL Polish

| PT Portuguese

| SK Slovak

| ES Spanish

| SV Swedish

| For testing of keyboard devices, additional low-level methods are defined for *"keyboard"* devices. **get-scancode** will read the next available scan-code from the keyboard; **scancode->char** will perform character mapping, using the same conversion as would normal **read**s. This enables test software to test for a specific character to terminate the test without having to know the scancode-to-character translations.

I.12.2.1 "Keyboard" Device Standard Properties

"device_type" Standard Open Firmware property; the value of this property for keyboard devices must be *"keyboard"*.

"language" Standard property, encoded as with **encode-string**, that indicates the current scan-code-to-character conversion based upon the language's keyboard layout.

The *"language"* standard property for keyboard devices is defined for PowerPC Open Firmware implementations. This value indicates the language (i.e. keyboard layout scan-code conversion) to which the keyboard driver is currently set.

I.12.2.2 "Keyboard" Device Methods

In addition to the Open Firmware standard **open, close** and **read** methods, the following methods must be supported by an Open Firmware implementation of a *"keyboard"* device. Note that the **open** routine can take an optional argument which specifies the language (i.e. **scancode->char** mapping) to be used.

get-scancode **(msecs -- scancode true | false)**

This method returns the "raw" scan-code value of the next key alteration. *msecs* is the number of milliseconds to wait for a keystroke before reporting failure; a value of 0 implies wait until keystroke. If this timeout expires before a keystroke is read, **false** is returned. Otherwise, the *scancode* and **true** are returned.

scancode->char **(scancode -- char true | false)**

Using the *scancode*, perform translation to a character. If the *scancode* represents a modifier key (e.g. Shift), no translation will be available; in this case, **false** is returned. If the *scancode* represents a translated character, *char* and **true** are returned.

set-language **(str len -- true | false)**

If the keyboard driver can support the requested character set, it must set its *"language"* property to the value specified by *str,len*, and return a value of **true**. If it can not support the requested language, no change of *"language"* is made and a value of **false** is returned.

The reason for adding a special call, instead of just using the **property** (*setprop*) call, is to allow the device to "filter" the request; i.e. a scan-code conversion may not be available for the requested language. This call allows the device driver to change the property only if it can support the requested mapping.

I.12.3 Pointing Devices

This class of device covers a broad category of "pointing" devices, the most common embodiment of which is the mouse. These devices can typically generate X,Y coordinates and button-press information on some periodic basis. The following properties and methods are defined for such devices.

I.12.3.1 "Mouse" Device Standard Properties

"device_type" Standard Open Firmware property; the value of this property for pointing devices must be *"mouse"*.

"#buttons" Standard property, encoded as with **encode-int**, that indicates the number of physical buttons supported by the device. This property can be used to interpret the *buttons* value returned by the **get-event** method.

"absolute-position" Standard property, whose presence indicates that this device supplies absolute X,Y coordinates (e.g. a graphics tablet). Absence of this property indicates that the device supplies relative X,Y position (e.g. a mouse).

I.12.3.2 "Mouse" Device Methods

In addition to the Open Firmware standard **open** and **close** methods, the following methods must be supported by an Open Firmware implementation of a *"mouse"* device.

Pointing devices typically supply data only when an "event" occurs (e.g. the mouse moves or a button is pressed). The following method attempts to obtain an event from the device, reporting whether an event occurred.

get-event **(msecs -- pos.x pos.y buttons true | false)**

Standard method for obtaining the next "event" of pointing devices. *msecs* is the number of milliseconds to wait for an event before reporting failure; a value of 0 implies wait until event. *pos.x*, *pos.y* return the positioning information; they are interpreted as unsigned or signed, depending upon the presence or absence of the *"absolute-position"* property. *buttons* returns a bit-mask (in the low-order bits) representing any buttons that are pressed; the number of significant bits to examine is defined by the *"#buttons"* property. The least significant bit corresponds to the leftmost button of the device. The top stack result indicates whether an event was detected within the time-out period.

I.12.4 Real-Time Clock (RTC) Device

The PowerPC Reference Platform requires the presence of a Real-Time Clock, with a minimum resolution of one second. Open Firmware for this clock defines the following properties and methods. The representation of time is defined by the TIME&DATE method of the ANS Forth standard [10].

I.12.4.1 "RTC" Device Standard Properties

"device_type" Standard Open Firmware property; the value of this property must be *"rtc"*.

I.12.4.2 "RTC" Device Methods

In addition to the Open Firmware standard **open** and **close** methods, the following methods must be supported by an Open Firmware implementation of an *"rtc"* device.

get-time **(-- n1 n2 n3 n4 n5 n6)**

Return the current time as the integers *n1...n6*, where *n1* is the second {0...59}, *n2* is the minute {0...59}, *n3* is the hour {0...23}, *n4* is the day {1...31}, *n5* is the month {1...12}, and *n6* is the year (e.g. 1994).

set-time **(n1 n2 n3 n4 n5 n6 --)**

Set the current time from the integers *n1...n6*, where *n1* is the second {0...59}, *n2* is the minute {0...59}, *n3* is the hour {0...23}, *n4* is the day {1...31}, *n5* is the month {1...12}, and *n6* is the year (e.g. 1994).

I.12.5 Sound Device

The PowerPC Reference Platform requires an audio subsystem with at least two channels for input and two channels for output, capable of sampling rates of at least 22.05 and 44.1 KHz to at least 16-bit resolution. In order to use this sound device within the context of Open Firmware (e.g. "boot beeps"), the following properties and methods must be implemented.

I.12.5.1 "Sound" Device Standard Properties

"device-type" Standard Open Firmware property; the value of this property must be *"sound"*.

"#channels" Standard property, encoded as with **encode-int**, that represents the number of channels supported by the device.

"sample-resolution" Standard property, encoded as with **encode-int**, that represents the number of bits of resolution for each sound sample.

"sample-width" Standard property, encoded as with **encode-int**, that represents the number of bytes required for storing a sample.

"sample-rates" Standard property, consisting of an array of integers, each encoded as with **encode-int**, that represents the rates (in hertz) at which this device can be sampled.

I.12.5.2 "Sound" Device Standard Methods

The following methods must be implemented by a *"sound"* device.

open **(-- true | false)**

This Standard method prepares the *"sound"* device for subsequent **read**s or **write**s. An argument can be supplied (i.e. following a ":" in the path-name component, available via **my-args**) to specify sampling parameters. The argument is a string consisting of the external representation of the sample-rate to be used; if absent, an implementation-defined sample rate is used. An error is signalled by **open** (i.e. it returns **false**) if the requested sample rate cannot be supported by the device.

close **(--)**

Standard Open Firmware behavior.

read **(addr size -- actual)**

Acquire sound data, storing the samples at *addr*. The sample rate is established by **open**.

write **(addr size -- actual)**

Output sound samples, stored at *addr*. The sample rate is established by **open**.

I.12.6 NVRAM Device

Access to the PowerPC Reference Platform-specific area of NVRAM is supported by this device-type. The NVRAM is treated as a device that can be read and written using the standard Open Firmware **read** and **write** methods; the starting position within the NVRAM can be specified by the **seek** method.

The PowerPC Reference Platform standard defines CRCs that are used to validate the integrity of the data within the NVRAM. To provide flexibility in using NVRAM, options are provided on the **open** method for NVRAM that determine whether the CRCs have to be valid at **open** time, and/or whether they are written at **close** time. These options appear as the argument component of the device specifier used to **open** the device (as with **open-dev**).

I.12.6.1 NVRAM Properties

"device-type" Standard Open Firmware property; the value of this property must be *"nvram"*.

I.12.6.2 NVRAM Methods

The following methods have the semantics of the Open Firmware methods:

read **(addr len -- actual)**

write **(addr len -- actual)**

seek **(pos.lo pos.hi -- status)**

The following methods have additional behavior depending upon the argument used to open the device:

open **(-- true | false)**

Standard method used to initiate access to the device and control how CRCs are used and/or generated. The **open** method must use **my-args** to determine special handling, as follows:

(no argument) **my-args** is not provided. The **open** method will verify the CRCs within the PowerPC Reference Platform's NVRAM area. If correct, the **open** will succeed, returning **true**; subsequent calls to **write** will cause the CRCs to be calculated and stored when the device is **close**'d. If not correct, **open** will report failure by returning **false**.

raw The **open** will succeed, without verifying the CRCs. Calls to **write** will not be reflected in the CRCs at **close** time.

repair The **open** will succeed, without verifying the CRCs. However, the CRCs will be regenerated at **close** time.

close **(--)** Standard method, whose behavior depends upon what the argument value was at the time the device was **open**'d. If *argument* was empty, or *repair*, the CRCs will be computed and stored.

I.12.7 Parallel Port Device

Access to the PowerPC Reference Platform parallel port is supported by this device type. The parallel port is treated as a byte-stream device.

I.12.7.1 Parallel Port Properties

"device-type" Standard Open Firmware property. The value of this property must be *"parallel"*.

I.12.7.2 Parallel Port Methods

The following methods have the semantics of the corresponding Open Firmware standard methods:

open **(-- true | false)**

close **(--)**

write **(addr len -- actual)**

I.12.8 Conventions for Devices on ISA and SCSI Buses

This section defines the naming and device type conventions for typical devices on ISA and SCSI buses. The following list shows the values of the “name” and “device-type” properties of the devices on an ISA bus:

```
name                    device_type
8042
        kbd             "keyboard"
        mouse           "mouse"
floppy                  "block"
com                     "serial"
timer                   "timer"
lpt                     "parallel"
ide                     "block"
nvram                   "nvram"
rtc                     "rtc"
```

Note: The “kbd” and “mouse” names are indented to show that they are the child nodes of the 8042 node.

Some systems use an I/O controller, often called a super I/O chip, which provides control functions of multiple I/O devices. When a system uses a super I/O chip, the device node of the super I/O chip must not be created. Instead, the device nodes of the devices attached to the super I/O chip must be implemented as a child node of the bus node to which the super I/O chip is attached.

The following list shows the values of the name and device-type properties of the devices on a SCSI bus:

```
name                device_type
scsi                "scsi"
     disk           "block"
     tape           "byte"
```

Note: The SCSI controller is considered a bus device in the device tree for PowerPC Open Firmware. The “disk” and “tape” names are indented to show that they are the child nodes of the “scsi” node.

If there are multiple instances of the same device type, for example two IDE hard disks or two SCSI hard disks, their names must be postfixed with indices to distinguish them, such as ide1 and ide2, or disk1 and disk2.

It is strongly recommended that the “compatible” property be implemented for ISA and SCSI bus devices to help operating systems find appropriate device drivers for these devices.

I.12.9 "/aliases" Node Properties

An implementation of Open Firmware for the PowerPC Reference Platform must provide the following aliases (for the preferred devices that exist) if applicable device exists under the “/aliases” node:

```
| disk
| tape
| cdrom
| keyboard
| screen
| scsi
| com1
| com2
| floppy
| net
```

Appendix J. Plug and Play Extensions

The structure below is the Plug and Play definition for PCI adaptors.

```
/* Structure map for PCI Bridge in PnP Vendor specific packet */

/* See Plug and Play ISA Specification, Version 1.0a, March 24, 1994.
   It (or later versions) is available on Compuserve in the PLUGPLAY
   area.  This code has extensions to that specification, namely new
   short and long tag types for platform dependent information */
/* Warning: LE notation used throughout this file */

#ifndef _PCIPNP_
#define _PCIPNP_

#define MAX_PCI_INTS    4

typedef enum _IntTypes {              /* interrupt controller types enumerator */
   IntCtl8259 = 1,                    /*    8259                               */
   IntCtlMPIC = 2                     /*    MPIC                               */
   } _IntTypes;

typedef struct _IntStruct {           /* PCI Int to IRQ conversion map */
   unsigned char  IntCtrlType;        /* Interrupt controller type */
   unsigned char  uchReserved[3];     /* Reserved (padded with 0)  */
   unsigned char  IntMask[4];         /* PCI INT mask              */
                                      /*   For 8259:               */
                                      /*       Bit 0 : INTA        */
                                      /*       Bit 1 : INTB        */
                                      /*       Bit 2 : INTC        */
                                      /*       Bit 3 : INTD        */
                                      /*       Others: Reserved    */
   unsigned char  IRQ[4];             /* corresponding IRQ number  */
                                      /*   to the IntMask bits set */
   } IntStruct;

typedef struct _IntMap {              /* PCI Int to 8259 or MPIC IRQ conversion map */
   unsigned char  SlotNumber;         /* Slot number engraved on the box */
                                      /*   zero indicates invalid entry   */
   unsigned char  DevFuncNumber;      /* PCI slot's DeviceFunction number */
                                      /*   zero indicates no option slot  */
                                      /*   non-zero indicates a valid option slot */
   unsigned char  uchReserved[2];     /* Reserved (padded with 0)    */
   IntStruct      isInt[MAX_PCI_INTS]; /* Interrupt mapping table     */
                                      /*   0, 0 indicates last valid */
                                      /*   entry. Entries after 0, 0 */
                                      /*   must be ignored           */
   } IntMap;

typedef struct _PCIInfoPack {
   unsigned char  Tag;                /* large tag = 0x84 Vendor specific  */
   unsigned char  Count0;             /* lo byte of count                  */
   unsigned char  Count1;             /* hi byte of count                  */
      /* count = number of pluggable PIC slots * sizeof(IntMap) + 6        */
   unsigned char  Type;               /* = 3 (PCI bridge)                  */
   unsigned char  ConfigBaseAddress[4];/* Base address of PCI Configuration */
   unsigned char  BusNumber;          /* PCI Bus number                    */
```

```
|    IntMap        Map[1];             /* Interrupt map array for each PCI  */
|                                      /*  slots that are pluggable         */
|    } PCIInfoPack;

| #endif  /* ndef _PCIPNP_ */
```

| The structure below is the Plug and Play definition for L2 cache devices.

```
| /* Structure map for L2 cache in PnP Vendor specific packet */

| /* See Plug and Play ISA Specification, Version 1.0a, March 24, 1994.
|    It (or later versions) is available on Compuserve in the PLUGPLAY
|    area.  This code has extensions to that specification, namely new
|    short and long tag types for platform dependent information */
| /* Warning: LE notation used throughout this file */

| #ifndef _L2PNP_
| #define _L2PNP_

| #define  L2Info_Packet  0x84          /* tag for L2Info_Pack  */

| typedef enum _L2_Store_Algorithm {
|    None = 0,
|    WriteThru = 1,
|    CopyBack = 2,
|    } L2_Store_Algorithm;

| typedef enum _L2_Cache_Asc {
|    DirectMapped = 1,                  /* direct mapped */
|    TwoWay = 2,                        /* 2-way         */
|    } L2_Cache_Asc;

| typedef enum _L2_HW_Assist {
|    Invalidate = 1,                    /* invalidate        */
|    Flush = 2,                         /* flush             */
|    L2PowerManaged = 4                 /* power managed     */
|    } L2_HW_Assist;

| typedef struct _L2InfoPack {
|    unsigned char  Tag;                /* large tag = 0x84 Vendor specific */
|    unsigned char  Count0;             /* = 0x0D sizeof(L2InfoPack - 3)    */
|    unsigned char  Count1;             /* = 0               */
|    unsigned char  Type;               /* = 2 (L2 cache)    */
|    unsigned char  L2_CacheSize[4];    /* In 1K bytes       */
|    unsigned char  L2_CacheAsc[2];     /* L2_Cache_Asc enum  */
|    unsigned char  L2_LineSize[2];
|    unsigned char  L2_SectorSize[2];
|    unsigned char  L2_StoreAlgorithm;  /* L2_Store_Algorithm enum */
|    unsigned char  L2_HWAssist;        /* L2_HW_Assist           */
|    } L2InfoPack;

| #endif  /* ndef _L2PNP_ */
```

The structure below is the Plug and Play definition for processor chip identities.

```
/* Structure map for Chip ID in PnP Vendor specific packet */

/* See Plug and Play ISA Specification, Version 1.0a, March 24, 1994.
   It (or later versions) is available on Compuserve in the PLUGPLAY
   area.  This code has extensions to that specification, namely new
   short and long tag types for platform dependent information */
/* Warning: LE notation used throughout this file */

#ifndef _CHPIDPNP_
#define _CHPIDPNP_

#define  ChipID_Packet     0x70        /* tag for ChipIDPack without size  */

typedef enum _Chip_ID {

   /* PCI Bridge chips */
   PCI_Br1 = 1,                        /* Part # IBM27-82650-653/4        */
   PCI_Br2 = 2,                        /* Part # IBM27-82657-50           */
   PCI_Br3 = 3,                        /* Part # MPC105                   */

   /* Power management chips (ISA) */
   PM_ISA_1 = 1,                      /*  Part # ---                      */
   Sig750 = 2,                        /*     Signetic 87C750              */

   /* Power management chips (PCI) */
   PM_PCI_1 = 1,                       /* Part # ---                     */

   /* L2 Cache controller */
   L2_Cntl_1 = 1,                     /*  Part # IBM27-82681-66        */
   L2_Cntl_2 = 2,                     /*  Part # for Energy managed WS */
   L2_Cntl_3 = 3,                     /*  Part # for Portable WS       */
   L2_Cntl_4 = 4,
   L2_Cntl_5 = 5,
   L2_Cntl_6 = 6,
   L2_Cntl_7 = 7

   } Chip_ID;

typedef struct _ChipIDPack {          /* This packet identifies chip set ID. */
                                      /* Details of these specifics are      */
                                      /* documented in the chip specs.       */
   unsigned char  Tag;                /* small tag = 0x7n with n bytes       */
   unsigned char  Type;               /*    = 1 (Chip ID)                    */
   unsigned char  VendorID0;          /*    Bit(7)=0                         */
                                      /*    Bits(6:2)=1st character in       */
                                      /*       compressed ASCII              */
                                      /*    Bits(1:0)=2nd character in       */
                                      /*       compressed ASCII bits(4:3)    */
   unsigned char  VendorID1;          /*    Bits(7:5)=2nd character in       */
                                      /*       compressed ASCII bits(2:0)    */
                                      /*    Bits(4:0)=3rd character in       */
                                      /*       compressed ASCII              */
   unsigned char  Name[2];            /*    Chip ID name                     */
   } ChipIDPack;

#endif  /* ndef _CHPIDPNP_ */
```

The structure below is the Plug and play definition for Disk drives.

```
/* Structure map for Diskette drives Vendor specific packet */

/* See Plug and Play ISA Specification, Version 1.0a, March 24, 1994.
   It (or later versions) is available on Compuserve in the PLUGPLAY
   area.  This code has extensions to that specification, namely new
   short and long tag types for platform dependent information */
/* Warning: LE notation used throughout this file */

#ifndef _DSKTPNP_
#define _DSKTPNP_

#define  Dskt_Packet 0x84               /* tag for DsktInfoPack          */

typedef struct _DsktInfoPack {
   unsigned char  Tag;                  /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;               /* lo byte of number of drives + 1  */
   unsigned char  Count1;               /* hi byte of number of drives + 1  */
            /* Count (Count0 and Count 1) - 1 = number of diskette drives    */
   unsigned char  Type;                 /*   = 1 (diskette)                 */
   unsigned char  Dskt[1];              /* diskette drives info array       */
                                        /*      0 : no drive present        */
                                        /*      1 : 3.5" 2MB drive          */
                                        /*      2 : 3.5" 4MB drive          */
                                        /*      3 : 5.25" 1.6MB drive       */
   } DsktInfoPack;

#endif  /* ndef _DSKTPNP_ */
```

Appendix K. Dump of Residual Data

A dump of the residual data constructed by a machine which matches the Reference Implementation is shown below:

```
Residual ID:
   Length = 0x6a0c
   Version = 0
   Revision = 0

Residual VPD:
   Model  = IBM PPS Model 6015
   Serial = ffffffffffffffff
   Spec Version  = 0
   Spec Revision = 0
   FW Support    = 0x1c5
   NVRAM size   = 4096
   SIMM slots   = 6
   ISA slots    = 3
   PCI slots    = 2
   PCMCIA slots = 0
   MCA slots    = 0
   EISA slots   = 0
   CPU MHz      = 66
   CPU bus MHz  = 66
   PCI bus MHz  = 33
   Time base div = 0xffffffff
   Word width = 32
   Page size  = 4096
   Block size = 32
   Granule size = 32
   Cache size   = 32
   Cache attrib = 2
   Cache assoc  = 8
   Cache line   = 64
   I-Cache size   = 32
   I-Cache assoc  = 8
   I-Cache line   = 64
   D-Cache size   = 32
   D-Cache assoc  = 8
   D-Cache line   = 64
   TLB size   = 256
   TLB attrib = 2
   TLB assoc  = 2
   I_TLB size   = 256
   I_TLB assoc  = 2
   D_TLB size   = 256
   D_TLB assoc  = 2
   ExtVPD ptr = 0x0

Residual CPU:
   Num CPUs = 1
     CPU[0] CPU Type   = 0x   10002
            CPU Serial = 0xffffffff
            L2 size    = 0
            L2 assoc   = 0
```

```
| Residual Memory:
|    Total Memory = 0x2000000
|    Good Memory  = 0x2000000
|    Mem seg[0] =
|      Mem seg is Firm Code
|      Mem seg base page  = 0x0
|      Mem seg page count = 0x400
|    Mem seg[1] =
|      Mem seg is Firm Heap
|      Mem seg base page  = 0x400000
|      Mem seg page count = 0x80
|    Mem seg[2] =
|      Mem seg is Free
|      Mem seg base page  = 0x480000
|      Mem seg page count = 0x1b80
|    Mem seg[3] =
|      Mem seg is Unpopulated
|      Mem seg base page  = 0x2000000
|      Mem seg page count = 0x7e000
|    Mem seg[4] =
|      Mem seg is SystemIO
|      Mem seg is ISAAddr
|      Mem seg base page  = 0x80000000
|      Mem seg page count = 0x800
|    Mem seg[5] =
|      Mem seg is SystemIO
|      Mem seg is PCIConfig
|      Mem seg base page  = 0x80800000
|      Mem seg page count = 0x800
|    Mem seg[6] =
|      Mem seg is SystemIO
|      Mem seg is PCIAddr
|      Mem seg base page  = 0x81000000
|      Mem seg page count = 0x3e800
|    Mem seg[7] =
|      Mem seg is SystemIO
|      Mem seg is SystemRegs
|      Mem seg base page  = 0xbf800000
|      Mem seg page count = 0x800
|    Mem seg[8] =
|      Mem seg is IOMemory
|      Mem seg base page  = 0xc0000000
|      Mem seg page count = 0x3f000
|    Mem seg[9] =
|      Mem seg is IOMemory
|      Mem seg base page  = 0xff000000
|      Mem seg page count = 0x800
|    Mem seg[10] =
|      Mem seg is UnPopSystemROM
|      Mem seg base page  = 0xff800000
|      Mem seg page count = 0x700
|    Mem seg[11] =
|      Mem seg is SystemROM
|      Mem seg base page  = 0xfff00000
|      Mem seg page count = 0x80
|    Mem seg[12] =
|      Mem seg is UnPopSystemROM
|      Mem seg base page  = 0xfff80000
|      Mem seg page count = 0x80
```

```
|     Mem seg[13] =
|       Mem seg is Boot Image
|       Mem seg base page  = 0x3b57e0
|       Mem seg page count = 0x4b
|     Total SIMMs = 4
|       SIMM[0] size = 8
|       SIMM[1] size = 8
|       SIMM[2] size = 8
|       SIMM[3] size = 8
|       SIMM[4] size = 0
|       SIMM[5] size = 0

|  Residual Devices:
|     Total Devices = 17
|     Device [0] = [PNPB00F] Crystal CS4231 Audio Device
|       Dev ID = 0x fb0d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 4
|       Sub   Type = 1
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 0
|       Possible   = 16
|       Compatible = 17
|     Device [1] = [PNP0700] PC Standard Floppy Disk Controller
|       Dev ID = 0x   7d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 1
|       Sub   Type = 2
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 18
|       Possible   = 29
|       Compatible = 30
|     Device [2] = [PNP0200] AT DMA Controller
|       Dev ID = 0x   2d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 8
|       Sub   Type = 1
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 31
|       Possible   = 152
|       Compatible = 153
```

```
|     Device [3] = [PNP0000] AT Interrupt Controller
|       Dev ID = 0x    d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 8
|       Sub   Type = 0
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 154
|       Possible   = 166
|       Compatible = 167
|     Device [4] = [PNP0A00] ISA Bus
|       Dev ID = 0x   ad041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 6
|       Sub   Type = 1
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 168
|       Possible   = 169
|       Compatible = 179
|     Device [5] = [PNP0400] Standard LPT Parallel Port
|       Dev ID = 0x   4d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Configurable
|       Device is Disableable
|       Device is Input
|       Device is Output
|       Base  Type = 7
|       Sub   Type = 1
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 180
|       Possible   = 188
|       Compatible = 214
|     Device [6] = [PNP0A03] PCI Bus
|       Dev ID = 0x 30ad041
|       Serial = 0xffffffff
|       Device is PCI
|       Device is Static
|       Base  Type = 6
|       Sub   Type = 4
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
```

```
|       Bus Access[2] = 0
|       Allocated  = 215
|       Possible   = 219
|       Compatible = 220
|     Device [7] = [PNP0B00] AT RTC
|       Dev ID = 0x   bd041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 8
|       Sub   Type = 3
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 221
|       Possible   = 229
|       Compatible = 230
|     Device [8] = [PNPA00F] NCR 810 SCSI Controller
|       Dev ID = 0x fa0d041
|       Serial = 0xffffffff
|       Device is PCI
|       Device is Configurable
|       Device is Disableable
|       Base  Type = 1
|       Sub   Type = 0
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 128
|       Bus Access[1] = 16
|       Bus Access[2] = 0
|       Allocated  = 231
|       Possible   = 235
|       Compatible = 236
|     Device [9] = [PNP0501] 16550A Compatible Serial port
|       Dev ID = 0x 105d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Configurable
|       Device is Disableable
|       Device is Input
|       Device is Output
|       Base  Type = 7
|       Sub   Type = 0
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 237
|       Possible   = 245
|       Compatible = 319
|     Device [10] = [PNP0501] 16550A Compatible Serial port
|       Dev ID = 0x 105d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Configurable
|       Device is Disableable
```

```
|       Device is Input
|       Device is Output
|       Base  Type = 7
|       Sub   Type = 0
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 320
|       Possible   = 328
|       Compatible = 402
|     Device [11] = [PNP0100] AT Timer
|       Dev ID = 0x   1d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 8
|       Sub   Type = 2
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 403
|       Possible   = 415
|       Compatible = 416
|     Device [12] = [PNP0303] IBM Enhanced (101/102 key, PS/2 mouse)
|       Dev ID = 0x 303d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Device is Disableable
|       Device is Removable
|       Device is ConsoleIn
|       Device is Input
|       Base  Type = 9
|       Sub   Type = 0
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 417
|       Possible   = 429
|       Compatible = 430
|     Device [13] = [PNP0F03] Microsoft PS/2 Mouse
|       Dev ID = 0x 30fd041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Device is Disableable
|       Device is Removable
|       Device is Input
|       Base  Type = 9
|       Sub   Type = 2
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
```

```
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 431
|       Possible   = 443
|       Compatible = 444
|     Device [14] = [PNP090E] Weitek P9000 Graphics Adapter
|       Dev ID = 0x e09d041
|       Serial = 0xffffffff
|       Device is PCI
|       Device is Configurable
|       Device is Disableable
|       Device is ConsoleOut
|       Base  Type = 3
|       Sub   Type = 0
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 6
|       Bus Access[2] = 0
|       Allocated  = 445
|       Possible   = 449
|       Compatible = 450
|     Device [15] = [PNP8327] IBM Token Ring (All types)
|       Dev ID = 0x2783d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Input
|       Device is Output
|       Base  Type = 2
|       Sub   Type = 1
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 451
|       Possible   = 480
|       Compatible = 481
|     Device [16] = [PNP8327] IBM Token Ring (All types)
|       Dev ID = 0x2783d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Input
|       Device is Output
|       Base  Type = 2
|       Sub   Type = 1
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 482
|       Possible   = 511
|       Compatible = 512

|  Residual Device Heap:

|    22   0   4  4b   8  30   4  2a  40  52
|    2a  80  52  71   1  78  78  78  22  40
```

```
|     0  4b   3  f0   8  2a   4  48  78  78
|    78  4b   0   0  10  4b   0  80  11  4b
|     0  94   3  4b   0  98   1  4b   0  9c
|     4  4b   0  c0   1  4b   0  c2   1  4b
|     0  c4   1  4b   0  c6   1  4b   0  c8
|     1  4b   0  ca   1  4b   0  cc   1  4b
|     0  ce   1  4b   0  d0   1  4b   0  d2
|     1  4b   0  d4   1  4b   0  d6   1  4b
|     0  d8   1  4b   0  da   1  4b   0  dc
|     1  4b   0  de   1  4b   4   b   1  4b
|     4  10   4  4b   4  15   7  4b   4  1d
|    13  4b   4  34   c  4b   4  81   3  4b
|     4  87   1  4b   4  89   3  4b   4  d6
|     1  78  78  78  22   4   0  4b   0  20
|     2  4b   0  a0   2  78  78  78  78  22
|     0   2  22   0   8  22   0  40  78  78
|    22  80   0  4b   3  bc   3  78  30  22
|    80   0  4b   3  bc   3  30  22  80   0
|    4b   3  78   6  30  22  20   0  4b   2
|    78   6  38  78  78  22   0  80  78  78
|    78  22   0   1  4b   0  70   2  78  78
|    78  22   0  20  78  78  78  22  10   0
|    4b   3  f8   8  78  30  22  10   0  4b
|     3  f8   8  30  22   8   0  4b   2  f8
|     8  30  22  10   0  4b   2  20   8  30
|    22  10   0  4b   2  e8   8  30  22  10
|     0  4b   3  38   8  30  22  10   0  4b
|     3  e8   8  30  22  10   0  4b   2  38
|     8  30  22  10   0  4b   2  e0   8  30
|    22  10   0  4b   2  28   8  38  78  78
|    22   8   0  4b   2  f8   8  78  30  22
|    10   0  4b   3  f8   8  30  22   8   0
|    4b   2  f8   8  30  22  10   0  4b   2
|    20   8  30  22  10   0  4b   2  e8   8
|    30  22  10   0  4b   3  38   8  30  22
|    10   0  4b   3  e8   8  30  22  10   0
|    4b   2  38   8  30  22  10   0  4b   2
|    e0   8  30  22  10   0  4b   2  28   8
|    38  78  78  22   1   0  4b   0  40   4
|    4b   0  78   4  78  78  78  22   2   0
|    4b   0  60   1  4b   0  64   1  78  78
|    78  22   0  10  4b   0  60   1  4b   0
|    64   1  78  78  78  22   0  80  78  78
|    78  4b   a  20   0  81   9   0  21   0
|     0   0   0   0  10   0   0  81   9   0
|     1   0   0   0   0   0  10   0   0  78
|    78  78  4b   0   0   0  81   9   0  21
|     0   0   0   0   0  10   0   0  81   9
|     0   1   0   0   0   0   0  10   0   0
|    78  78  78   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
```

Obtaining Additional Information

Several sources exist for obtaining additional information about the PowerPC microprocessor and the PowerPC Reference Platform.

Anyone interested in obtaining more information on the PowerPC processor or on the PowerPC Reference Platform may use the following sources:

- IBM at 1-800-PowerPC (1-800-769-3772) in the U.S.A (MPR-PPC-RPU-02 is the number for the *PowerPC Reference Platform Specification*, Version 1.0)
- If the 1-800-PowerPC number can not be reached or if multilingual operators are required, use 1-708-296-6767
- IBM at (39)-39-600-4295 in Europe
- Motorola at 1-800-845-MOTO (6686)

Copies of the *PowerPC Reference Platform Specification* may be obtained from these numbers. Hardware system vendors may obtain information on IBM components or the IBM design kits which give further information on the reference implementation by contacting IBM at the numbers listed above or at the following numbers:

- Within Europe (33)-6713-5757 in French
- Within Europe (33)-6713-5756 in Italian
- Within Europe (49)-511-516-3444 in English
- Within Europe (49)-511-516-3555 in German
- In Asia (81)-755-87-4745 in Japanese

An electronic forum on CompuServe has been established for the discussion of PowerPC Reference Platform topics and for obtaining answers to questions on the PowerPC Reference Platform. Go to the "PowerPC" forum on CompuServe and join the "Reference Platform" topic. The *PowerPC Reference Platform Specification* in PostScript format is maintained on a library of the PowerPC forum.

Several white papers are available from IBM at 1-512-838-5552. These papers expand on the information in this specification. The papers which were available at the time of publishing this specification are as follows:

- *PowerPC 60x/PCI Bus Bridge Implementation for PowerPC Reference Platform* by Yongjae Rim
- *Bi-Endian Designs in PowerPC Reference Platform* by Shien-Tai Pan
- *Symmetric Multiprocessing* by Don McCauley
- *PowerPC Endian Switch Code* by Gary Tsao
- *Plug and Play for PowerPC Reference Platform* by Gary Tsao
- *L2 Cache Design for PowerPC* by Allan Steel

This specification, the white papers, and various data structures (e.g. NVRAM Header) are available via anonymous FTP. The server address is ftp.austin.ibm.com and the material is placed in the directory /pub/technology/spec.

A paper, *The PowerPC Reference Platform Specification and Machine Abstraction* by Allan Steel, is available in the Spring Compcon 94 digest.

For AIX license and product information, contact David Hall, AIX OEM Relations, at 512-838-2088.

Software vendors interested in porting their applications to AIX on PowerPC systems can contact the AIX Power Team General Information Line at 1-800-222-2363.

A Windows NT porting center is available to help vendors who are interested in porting their products to this platform. The center can be contacted by telephone at 1-800-803-0110 or 1-206-889-9011, or by electronic mail on Internet at winntppc@vnet.ibm.com.

The PowerPC Architecture, ISBN 1-55860-316-6, is published by Morgan Kaufmann Publishers, 415-392-2665. It is available in some bookstores and may be ordered from local IBM publications sources at 1-800-426-6477 in the US only.

Kernel Extensions and Device Support Programming Concepts, IBM document number SC232207, and *Writing a Device Driver for AIX Version 3.2*, IBM document number GG243629, may be ordered by calling 1-800-879-2755.

/ To purchase a copy of *System V Application Binary Interface*, call 1-800-947-7700 in the U.S.A. or 515-284-6751
/ outside the U.S.A.

/ To order the *Guide to Mac Family Hardware*, call 1-800-282-2732.

/ Back issues of the *Microsoft Systems Journal* may be ordered by calling 1-800-444-4881 in the U.S.A. or 415-715-6213
/ outside the U.S.A.

/ To purchase the *MS-DOS Programmer's Reference*, call 1-800-677-7377.

| Call 212-642-4900 for orders or inquiries pertaining to ANSI and ISO standards.

To order copies of EIA standards, contact Global Engineering Documents at 1-800-854-7179.

Copies of IEEE standards may be obtained by calling 1-800-678-IEEE. For information about IEEE standards, call 908-562-3800.

/ To purchase PCI documents, call 1-800-433-5177 in the U.S.A. or 503-797-4207 outside the U.S.A.

Copies of PCMCIA standards may be obtained by calling 408-720-0107.

The PowerOpen Association may be contacted at the address and telephone number listed below:

PowerOpen Association
10500 North Wolfe Road
Suite SW2-255
Cupertino, CA 95014
1-800-457-0463

Bibliography

Documents which were referenced in this specification are listed below:

- *The PowerPC Architecture*, ISBN 1-55860-316-6
- *Kernel Extensions and Device Support Programming Concepts*, IBM order number SC232207
- *Writing a Device Driver for AIX Version 3.2*, IBM order number GG243629
- *How To Create Endian-Neutral Software for Portability*, TR54.837 (also published in *Dr. Dobb's Journal* for October and November 1994)
- PowerPC processor-specific user's manuals:
  - *PowerPC 601 RISC Microprocessor User's Manual*, IBM order number 52G7484, Motorola order number MPC601UM/ADREV1
- *ANSI Standard X3.131-1990 (Revision 10c) for SCSI-2*
- *ANSI Standard X3.221 AT Attachment (Revision 4a)*
- *ANSI/NISO/ISO 9660, Information Processing -- Volume and File Structure of CD-ROM for Information Interchange*
- EIA/TIA-232-E, *Interface Between Data Terminal Equipment and Data Circuit Terminated Equipment Employing Serial Binary Data Interchange*
- EIA-422-A, *Electrical Characteristics of Balanced Voltage Digital Interface Circuits*
- IEEE P1275, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices*
- IEEE P1284, *Standard Signaling Method for a Bi-Directional Parallel Peripheral Interface for Personal Computers*, March 15, 1993
- IEEE 802.3, ISO/IEC 8802-3, *Information technology -- Local and metropolitan area networks Part III: Carrier sense multiple access with collision detection (CSMA-CD) access method and physical layer specifications*, 1993
- IEEE 802.5, *Standards for Local Area Networks: Token Ring Access Method and Physical Layer Specifications*
- IEEE 996, *A Standard for an Extended Personal Computer Back Plane Bus*
- *ISO 639, Code for the representation of names of languages*
- *Microsoft Hardware Abstraction Layer*, Beta Version, March 1993
- PCI documents (available from and maintained by the PCI Special Interest Group):
  - *PCI Local Bus Specification*, Revision 2.0, April 30, 1993
  - *PCI System Design Guide*, Revision 1.0, September 1993
  - *PCI to PCI Bridge Architecture Specification*, Revision 1.0, April 5, 1994
  - *PCI Multimedia Design Guide*, Revision 1.0, March 29, 1994
- PCMCIA standards (can be ordered from the Personal Computer Memory Card International Association):
  - *PC Card Standard Specification*, Release 2.1, July 1993
  - *Socket Services Specification*, Release 2.1, July 1993
  - *Card Services Specification*, Release 2.1, July 1993
  - *PC Card ATA Mass Storage Specification*, Release 1.02, July 1993
  - *AIMS Specification*, Release 1.01, November 1992
  - *Recommended Extensions*, Release 1.00, November 1992
- Michel Dubois, Christoph Scheurich, and Faye Briggs, *Memory Access Buffering in Multiprocessors*, Proceedings of the 13th ISCA, pp. 434-441, 1986.
- *PowerOpen ABI*
- *PowerOpen API*
- *System V Application Binary Interface*
- *MS-DOS Programmer's Reference*

- “Peering Inside the PE: A Tour of the Win32 Portable Executable File Format,” *Microsoft Systems Journal*, March 1994
- *Bootstrap Protocol*, Internet RFC 951.

Acronyms and Abbreviations

Table 40 (Page 1 of 4). Acronyms and Abbreviations

Term	Definition
ABI	Application binary interface
ANSI	American National Standards Institute
API	Applications programming interface
APM	Available processor mask
ASIC	Application-specific integrated circuit
ATM	Asynchronous transfer mode
BAT	Block address translation
BE	Big-Endian
BEPI	Block effective page index
BL	Block length
BLR	Branch to link register
BRPN	Block real page number
BTAS	Boot-time abstraction software
BUID	Bus unit identifier
CB	Copy-back
CISC	Complex instruction set computer
CPPR	Current processor priority register
CRC	Cyclic redundancy check
CV	Compatible value
DAC	Digital-to-analog converter
DBAT	Data block address translation register
DCE	Distributed computing environment
DDI	Device driver interface
DDK	Device driver development kit
DMA	Direct memory access
DRAM	Dynamic random access memory
DSI	Data storage interrupt
DSP	Digital signal processor
EA	Effective address
ECC	Error checking and correcting
ECP	Extended capabilities port
EEPROM	Electrically erasable programmable read-only memory
EISA	Extended industry standard architecture
ELF	Executable and linking format
EOI	End of interrupt
EPLD	Electrically programmable logic device
EPROM	Erasable programmable read-only memory

Table 40 (Page 2 of 4). Acronyms and Abbreviations	
Term	**Definition**
FAL	Firmware abstraction layer
FAT	File allocation table
FCS	Fiber Channel Standard
FDDI	Fiber distributed data interface
FIFO	First in first out
FRU	Field-replaceable unit
GB	Gigabyte
GMT	Greenwich mean time
GOT	Global offset table
GPR	General purpose register
GUI	Graphical user interface
HAL	Hardware abstraction layer
HAS	Hardware abstraction software
HPFS	High-performance file system
HTAB	Hashed page table
IBAT	Instruction block address translation register
IC	Integrated circuit
IDE	Integrated device electronics
IP	Interrupt prefix
IPIRR	Inter-processor interrupt request register
IRDT	Interrupt redirection table
IRQL	Interrupt request level
IRQP	Interrupt request priority
IRR	Interrupt request register
ISA	Industry standard architecture
ISBN	International standard book number
ISDN	Integrated services digital network
ISE	Instruction set emulator
ISI	Instruction storage interrupt
ISR	Interrupt source register
ITL	Independent test lab
JEIDA	Japan Electronic Industry Development Association
JFS	Journaled file system
KB	Kilobyte
KBI	Kernel binary interface
KPI	Kernel programming interface
L1	First-level cache
L2	Second-level cache
LAN	Local area network
LBX	Local branch exchange

Table 40 (Page 3 of 4). Acronyms and Abbreviations

Term	Definition
LE	Little-Endian
LSb	Least significant bit
LSB	Least significant byte
LVM	Logical volume manager
MB	Megabyte
MCA	Micro Channel Architecture
MIDI	Musical instrument digital interface
MK	Microkernel
MMU	Memory management unit
MP	Multiprocessor
MSb	Most significant bit
MSB	Most significant byte
MSR	Machine status register
NFS	Network file system
NVRAM	Non-volatile random access memory
OEM	Original equipment manufacturer
ONC	Open network computing
OS	Operating system
PCI	Peripheral component interconnect
PCIB/MC	PCI bridge and memory controller
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal digital assistant
PE	Portable executable
PIM	Platform-independent module
PLL	Phase lock loop
PnP	Plug and Play
POE	PowerOpen Environment
POST	Power-on self test
PSM	Platform-specific module
PTE	Page table entry
QFP	Quad flat pack
QIC	Quarter-inch cartridge
RA	Real address
RAID	Redundant array of independent disks
RAMDAC	Random access memory and digital-to-analog converter
RBA	Relative block address
RFC	Request for comments
RGB	Red-green-blue
RISC	Reduced instruction set computer
RPN	Real page number

Table 40 (Page 4 of 4). Acronyms and Abbreviations	
Term	**Definition**
RTAS	Run-time abstraction software
SCSI	Small computer system interface
SIMM	Single inline memory module
SIO	Standard I/O
SMP	Symmetric multiprocessor
SODIMM	Small outline dual inline memory module
SPARC	Scalable processor architecture
SPR	Special purpose register
SR	Segment register
SSD	Storage system division
SVID	System V interface definition
TCP/IP	Transmission control protocol internet protocol
TEA	Transaction error acknowlegement
TLB	Translation lookaside buffer
TTY	Teletypewriter
UMCU	Universal micro control unit
VA	Virtual address
VL	Network VESA local bus
VME	VERSA Module Eurocard
VPD	Vital product data
VPN	Virtual page number
VSID	Virtual segment ID
WT	Write-through

Glossary

Alpha-numeric input device. A device for the input of symbols and alphabetical and numeric characters, such as a keyboard.

BAT register. A mechanism which provides a means of mapping ranges of virtual addresses larger than a page onto contiguous areas of real storage.

Big-Endian. A byte ordering method in which the most significant byte is stored first.

Bi-Endian. Having Big-Endian and Little-Endian byte ordering capability.

Dataless. A hardware configuration in which the system's hardfile is used only for system support; most of the system software and data reside on a network-connected storage device.

Directly attached. Electrically connected.

/ **Hard disk**. A storage media consisting of several spinning platters on which information is read and written electron-
/ ically.

/ **Hardfile**. A storage media for reading and writing large volumes of data. Typically implemented as a hard disk, but
/ other storage technologies may be used.

HUMAN-CENTERED. Technology which is centered around the human senses of sight, sound and touch.

Hot-plug capability. The ability to couple a peripheral device to a system without shutting down or restarting the system.

Little-Endian. A byte ordering method in which the least significant byte is stored first.

Medialess. A hardware configuration with no data storage capability. A network connection supplies storage for the operating system, applications and data.

Microkernel. A small, message-passing nucleus of system software running in the most privileged state of the computer.

Pointing device. A device which provides two-dimensional positioning, such as a mouse, tablet or touch screen.

Trademark Information

The following terms, denoted by a double asterisk (**) in this publication, are trademarks or registered trademarks of the companies shown:

/	DeskSet	Sun Microsystems, Inc.
	EtherCard	Standard MicroSystems Corporation
	EtherExpress	Intel Corporation
	EtherLink	3Com Corporation
	Ethernet	Xerox
	LocalTalk	Apple Computer, Inc.
	Macintosh	Apple Computer, Inc.
	NetWare	Novell, Inc.
/	Newscard	Motorola, Inc.
/	NFS	Sun Microsystems, Inc.
/	ONC	Sun Microsystems, Inc.
/	OpenWindows	Sun Microsystems, Inc.
	PostScript	Adobe Systems Incorporated
	SatisFAXtion	Intel Corporation
/	ScanJet	Hewlett-Packard
	Solaris	Sun Microsystems, Inc.
	Sound Blaster	Creative Technology, Ltd.
/	SunOS	Sun Microsystems, Inc.
/	ToolTalk	Sun Microsystems, Inc.
|	UNIX	X/Open Company, Ltd.
/	Wabi	Sun Microsystems, Inc.
	WangDAT	WangDAT Inc.
	Windows	Microsoft Corporation
	Windows NT	Microsoft Corporation
	X Window System	Massachusetts Institute of Technology

Index

I

K

L

M

N

O

P

R

S

T

U

V

W

END OF DOCUMENT

This is the last page of this document