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.

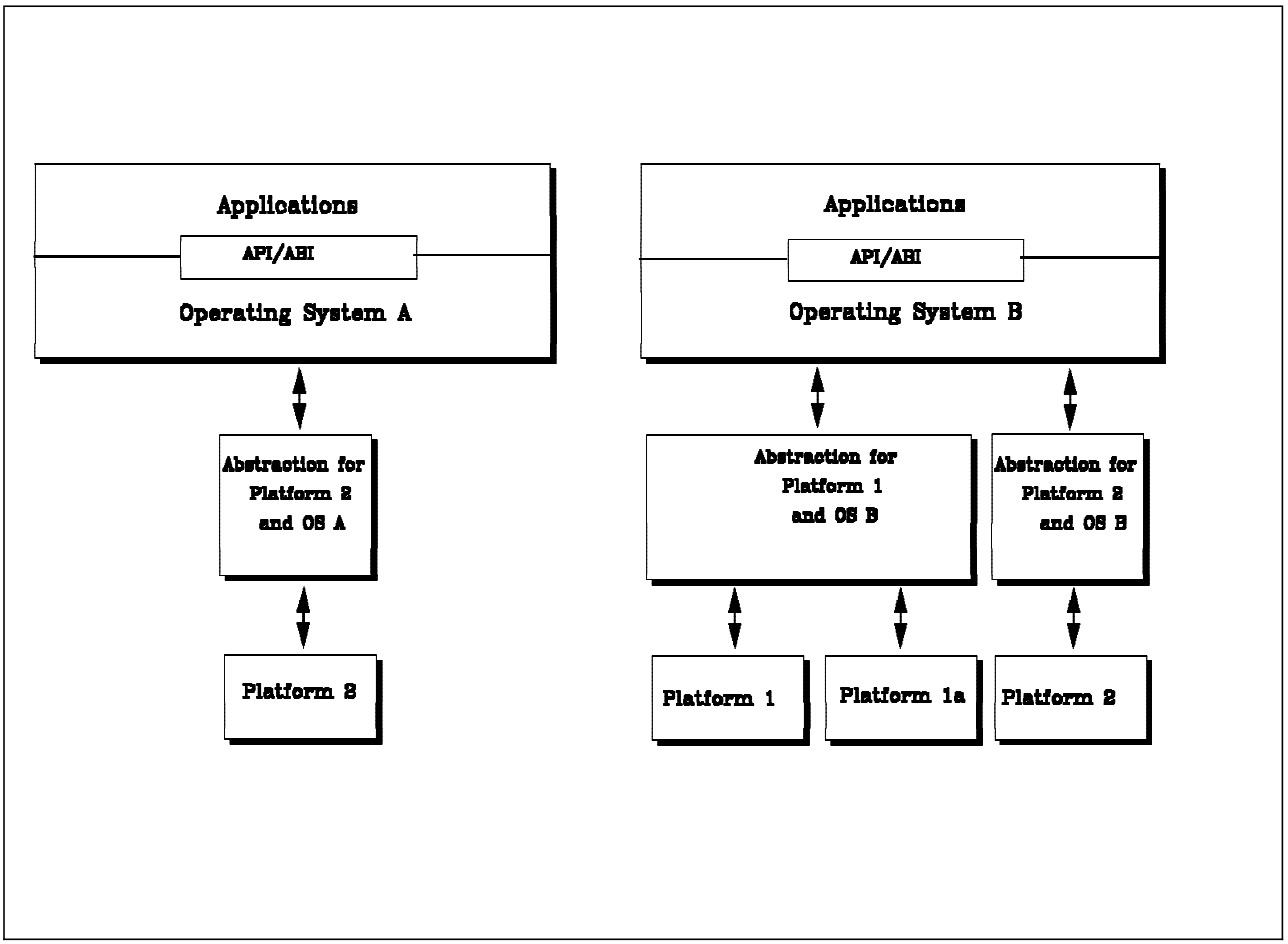


Figure 9. Abstraction Software for Various Platforms

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

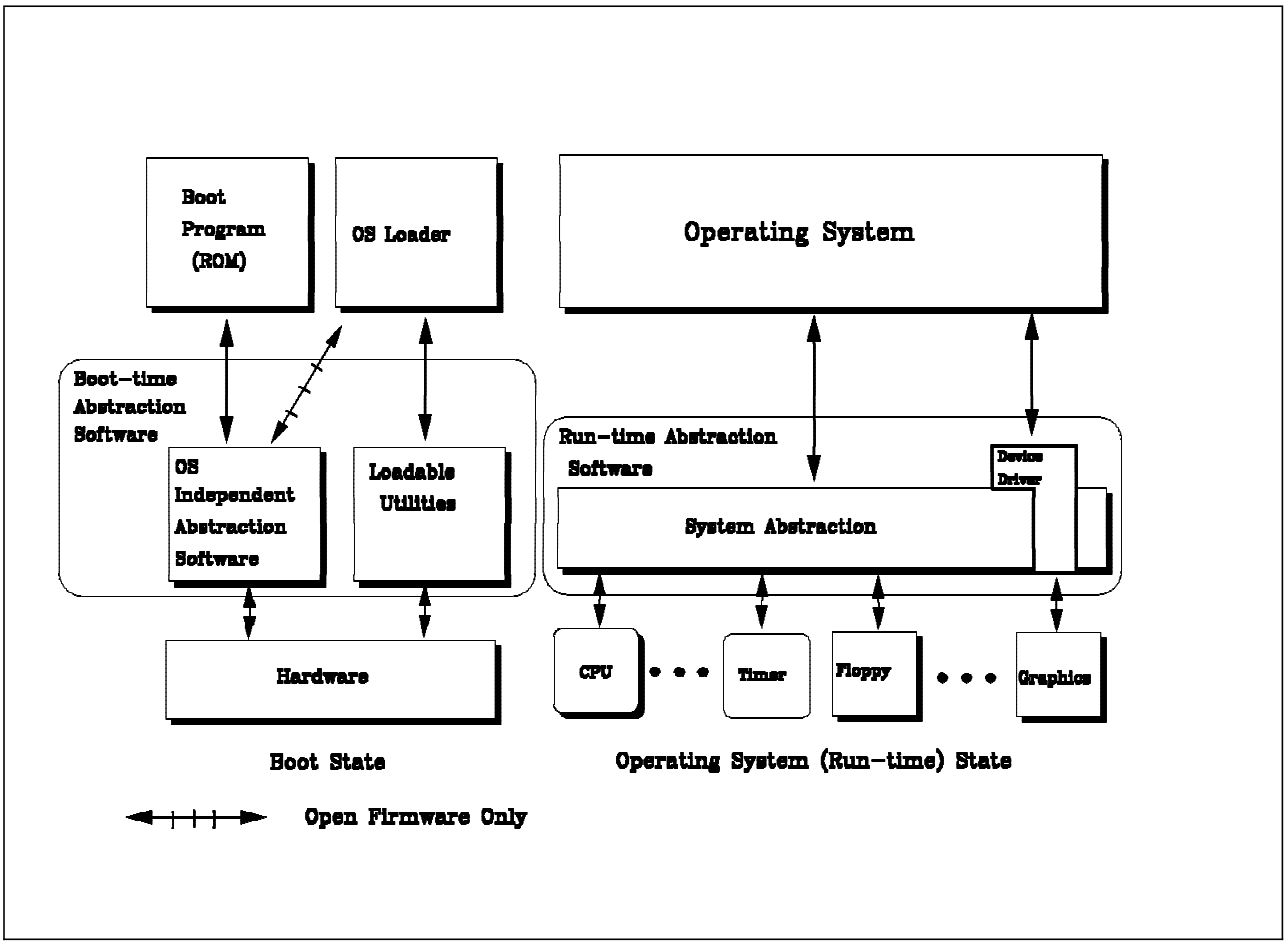


Figure 10. Software Abstraction Layers

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.

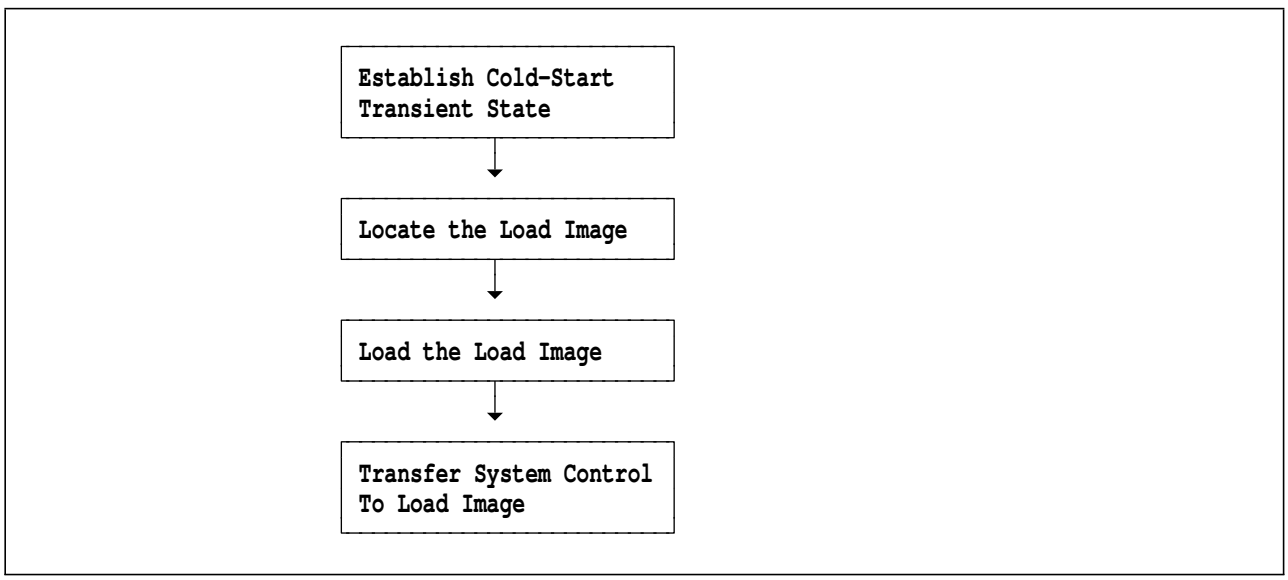


Figure 11. Boot Process Overview

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.

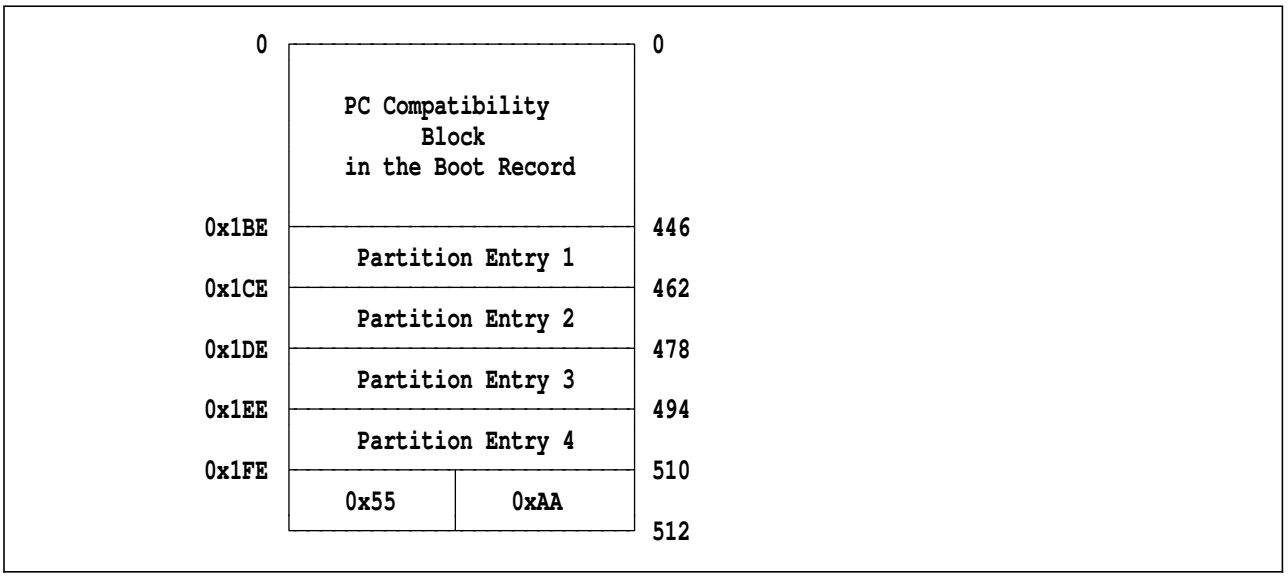


Figure 12. Boot Record -- Detail View

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.

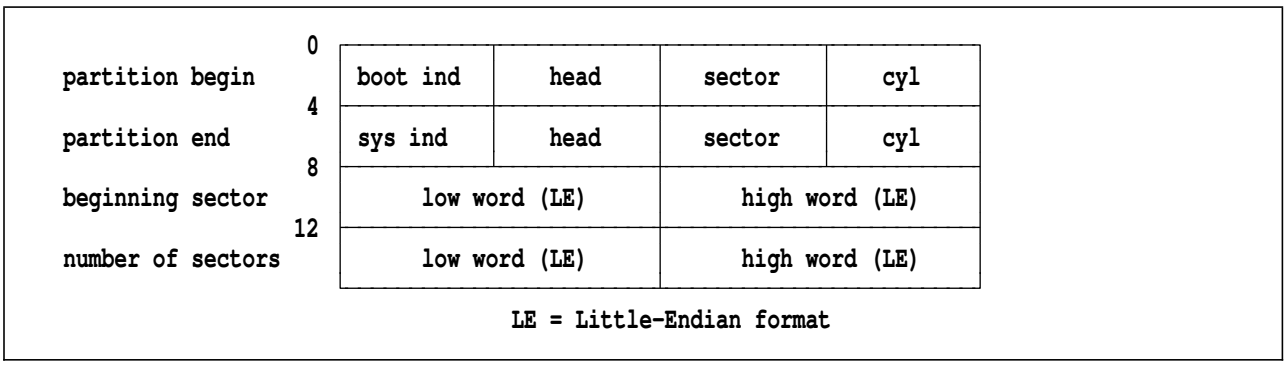


Figure 13. Partition Table Entry

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.

partition begin	boot ind	-1	-1	-1
partition end	sys ind	-1	-1	-1
beginning sector	32-bit start RBA (zero-based) (LE)			
number of sectors	32-bit RBA count (one-based) (LE)			

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.

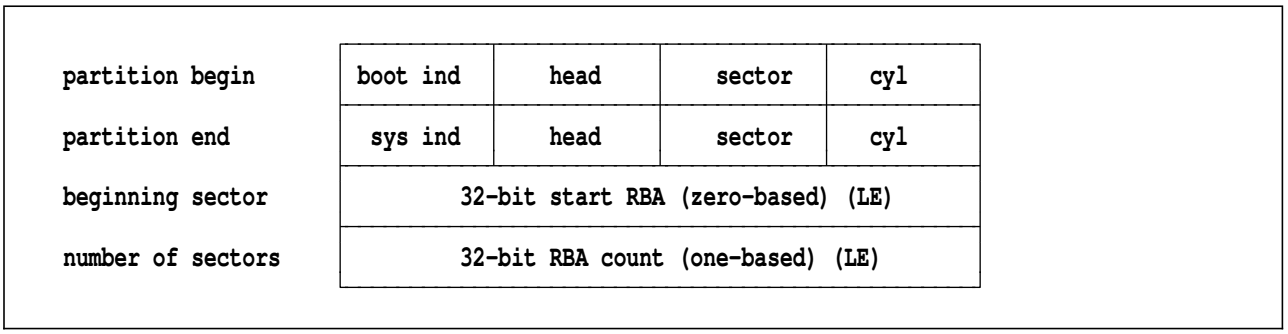


Figure 15. PowerPC Reference Platform Partition Table Entry Format for Conventional Firmware

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.

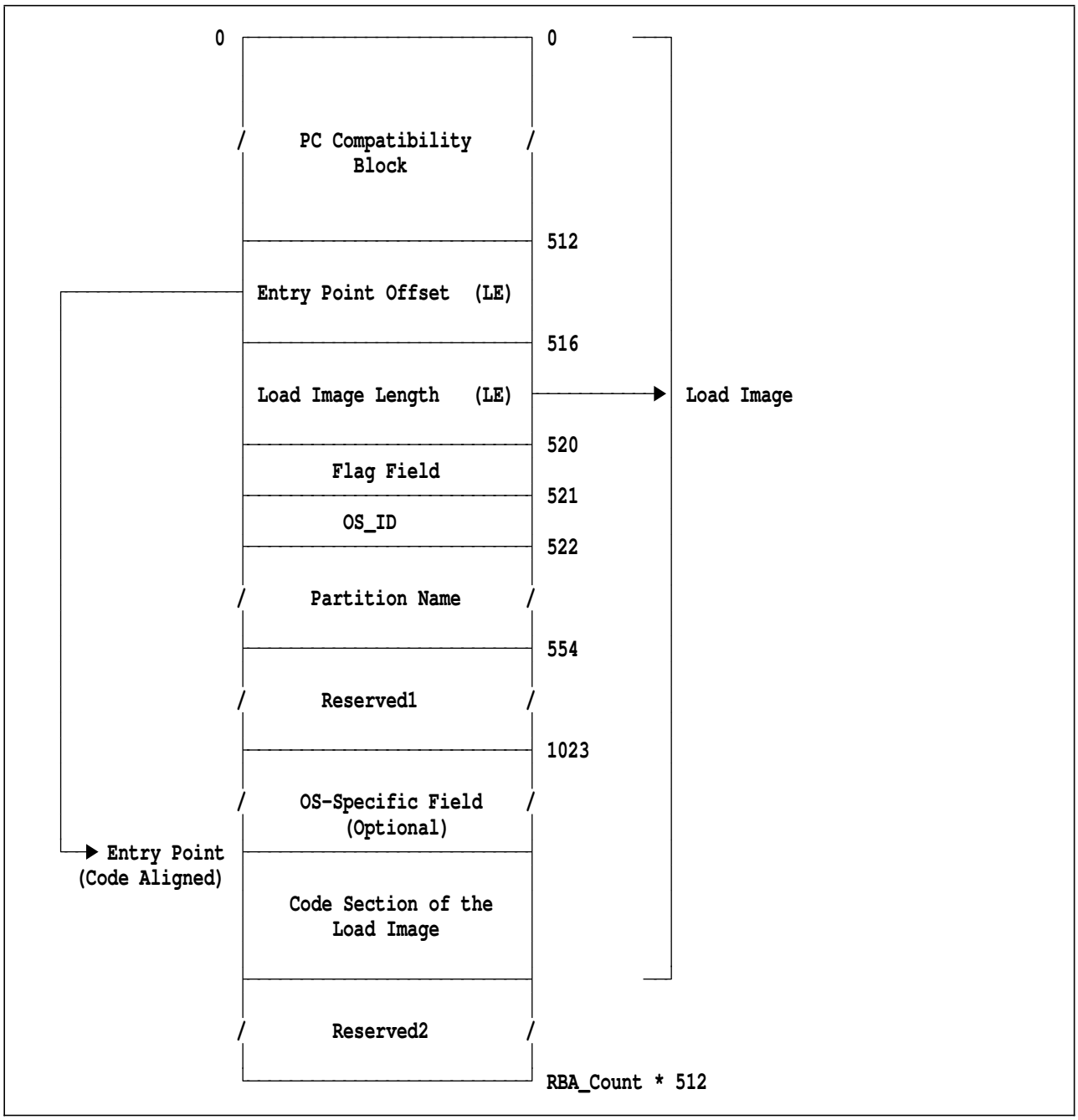
/ The 32-bit value load image length (Little-Endian) is the length, in bytes, of the load image. The load image / length specifies the size of the data physically copied into the system RAM by the firmware.

/ 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.

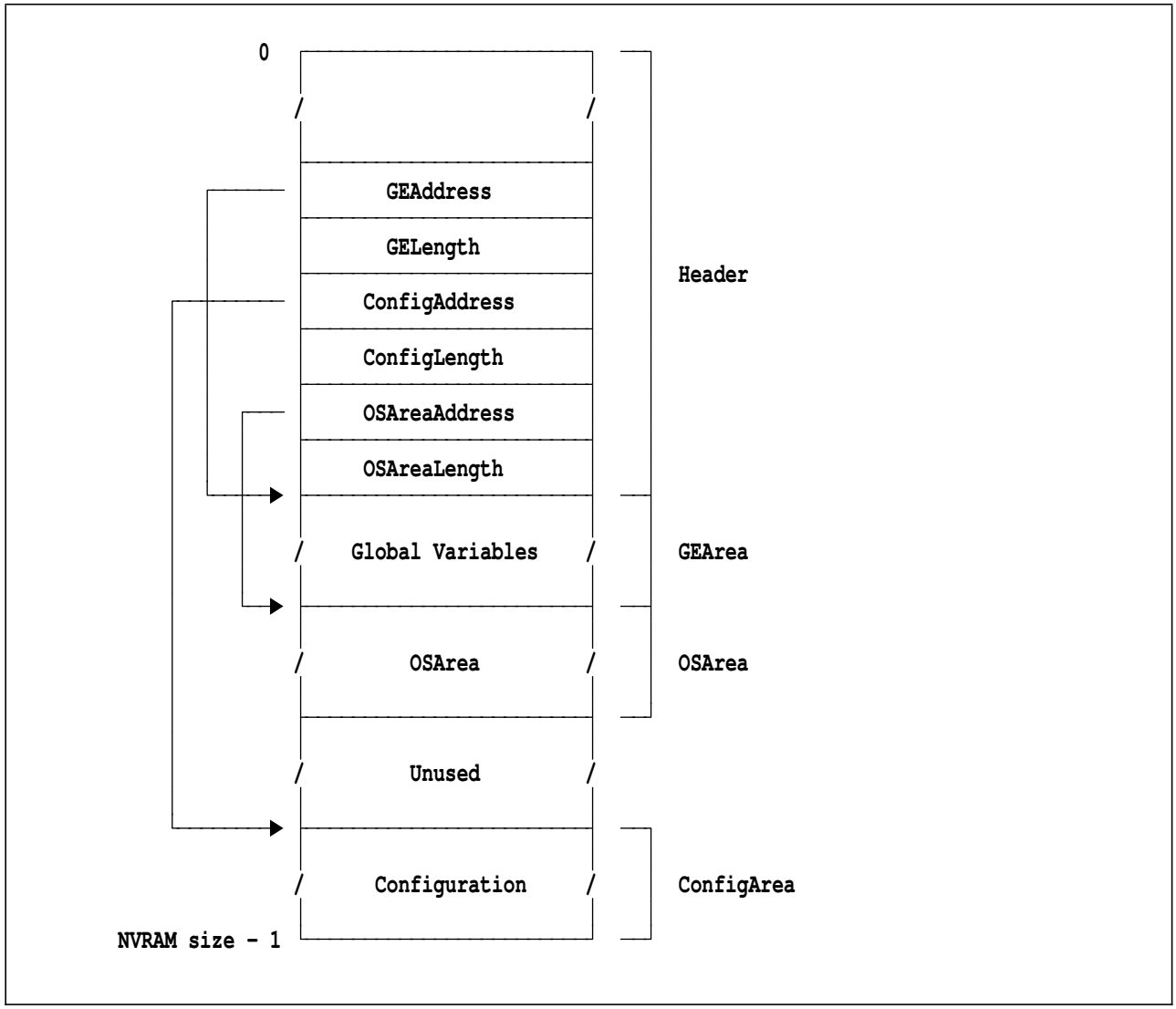
5.4.2 Call-Back to Firmware

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.

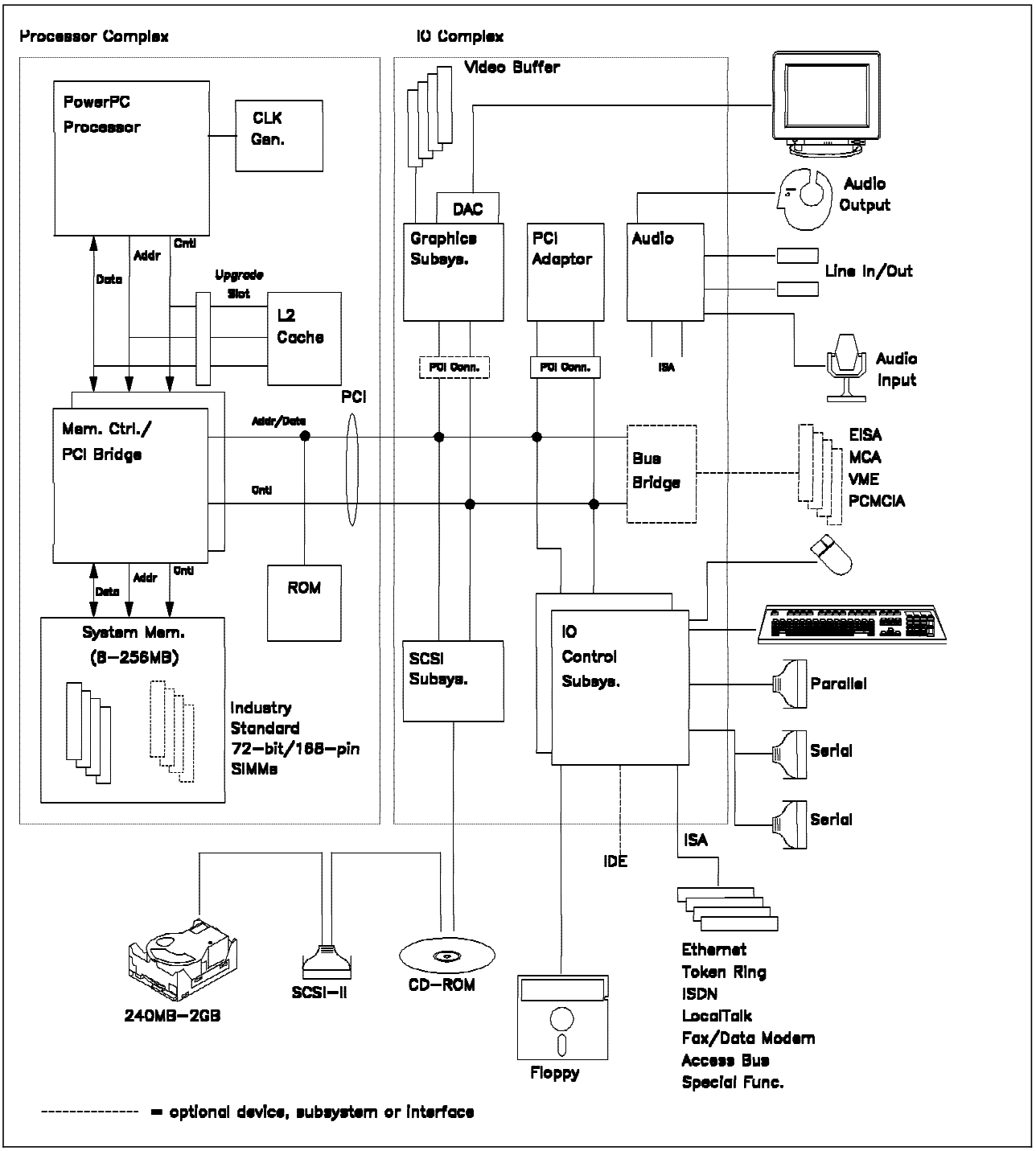


Figure 18. PowerPC Reference Platform Recommended Desktop System

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.