request is in response to a snoop request, the memory controller/PCI host bridge (or the L2 cache) must compare the address for the snoop request with the address for the cache write block request.

- The CPU-to-PCI posted write buffers free up the processor and the system bus for other activity while the PCI write operation proceeds independently. Typical configurations include:
  - Four 8-byte CPU-to-PCI write buffers.
  - Two 16-byte CPU-to-PCI write buffers.
  - One 32-byte CPU-to-PCI write buffer.

Snoop requests which hit on any of the CPU or PCI posted write buffers are normally retried.

- The read prefetch buffer, typically one 32-byte buffer, is used to hold data read from main memory for the last PCI read request. Subsequent PCI read requests to the same 32-byte sector access the read prefetch buffer, freeing up main memory for other requests. The read prefetch buffer is invalidated if a CPU or PCI write request hits on the buffer.
- The System Bus Interface Unit typically includes arbitration logic for three system bus masters (two in-line L2 caches plus the memory controller itself). Some systems, such as a four-way SMP, include arbitration for up to five system bus masters.
- The PCI Bus Interface Unit typically includes arbitration logic for six PCI bus masters (including the PCI host bridge itself) per PCI bus.

PCI (bus master) requests to and from System Memory must be maintained coherent. These requests typically cause a snoop operation on the system bus. PCI read requests are typically retried if the snoop request hits on a modified line in any processor's cache. PCI write requests are typically retried if the snoop request hits on a line in any processor's cache.

Sync and *eieio* requests from the system bus cause the memory controller/PCI host bridge to flush all posted write buffers.

The PowerPC system bus typically runs synchronous with the PCI bus at one, one and one-half, two, or three times (programmable) the PCI bus frequency. Some systems run the PowerPC system bus asynchronous with respect to the PCI bus; these systems support a wider variety of internal processor, processor bus and system bus frequencies at some increase in cost and complexity.

Performance observations for the memory controller/PCI host bridge:

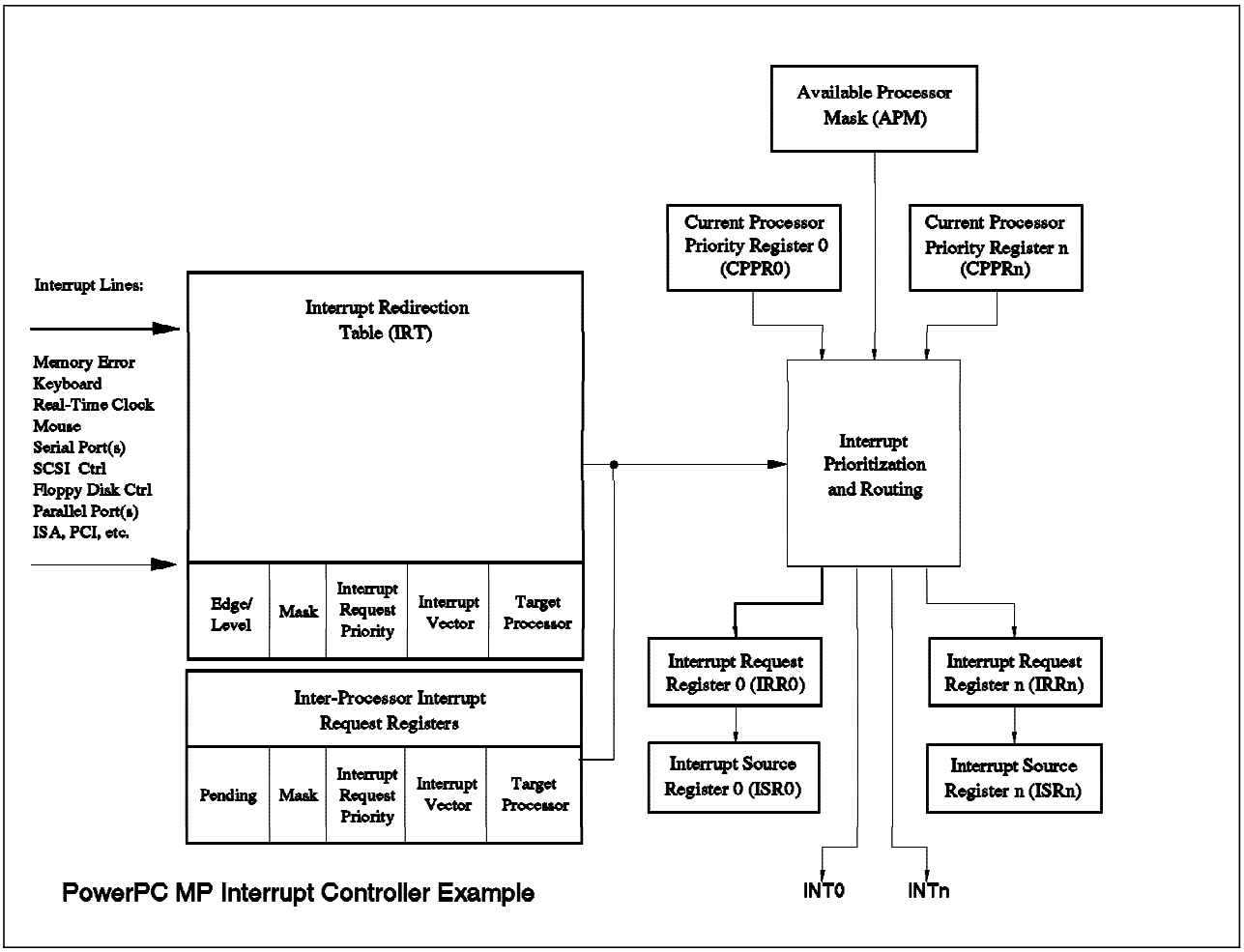
- Main memory read access times are normally stated in terms of number of cycles from transfer start (TS#) on the PowerPC system bus to the first data beat (DH, DL) and number of cycles between data beats. For systems using 70 ns synchronous DRAM (preferred), 7-1-1-1 is a typical main memory read access time (at 66 MHz). For systems using 70 ns asynchronous DRAM, 7-3-3-3 is a typical main memory read access time. (Write times, excluding the effect of the posted write buffers, are similar.)

A.6.4 MP Interrupt Controller Example A

Figure 52 is an example of a feature-rich MP interrupt controller. (Many other implementations are possible.) Features include:

One entry in the Interrupt Redirection Table (IRT) for each I/O interrupt. Each entry in the IRT includes:

- Edge or Level: 1 bit. (Edge-sensitive interrupts are converted, internally, to level-sensitive interrupts.)
  - 0 = edge-sensitive
  - 1 = level-sensitive
- Mask: 1 bit.
  - 0 enables the interrupt
  - 1 masks the interrupt



/ Figure 52. MP Interrupt Controller

/ • Interrupt Request Priority (IRQP): Typically 5 to 8 bits (32 to 256 interrupt priority levels). This is the
/ priority assigned, by software, to this interrupt. This field is optional; interrupt request priority can be
/ implemented in hardware or interrupt priority can be resolved through a software routine.
/ • Interrupt Vector (IRV): Typically 8 bits (256 interrupt vectors). This is the interrupt vector assigned,
/ by software, to this interrupt.
/ • Target Processor: Typically 2 to 8 bits. The interrupt is normally routed to the specified target
/ processor. A target processor of all ones (e.g. 'FF'X) enables priority-based interrupt routing (see
/ below).

/ The Inter-Processor Interrupt Request Registers (IPIRRs) are used for inter-processor interrupts. The field
/ definitions in the IPIRR are the same as for the Interrupt Redirection Table except that the Pending bit
/ replaces the Edge/Level bit.

/ The IPIRR can be used with the Target Processor field set (by software) to all ones to generate a floating
/ system-wide interrupt request. The IPIRRs allow software to maintain system-wide interrupt queue(s) (one
/ or more per system) in addition to processor-specific interrupt queues (one or more per processor).

/ The Target Processor fields in the IRT and in the IPIRR are used to redirect an interrupt request to a spe-
/ cific processor or to all processors.

/ The Current Processor Priority Registers (CPPR - one per processor) are used to assign a priority level to
/ each processor. These registers are required only if priority-based interrupt routing is implemented.

/ For interrupts targeted to a specific processor, an interrupt request with an IRQP higher than that
/ processor's current priority interrupts that processor; interrupt requests with an IRQP equal to or less than
/ that processor's current priority remain pending until that processor's priority is lowered.

/ For interrupts using priority-based interrupt routing, only one processor (normally the first processor whose
/ priority (CPPR) drops below the interrupt request's priority (IRQP)) is interrupted. In the case of a tie, one
/ processor is arbitrarily selected.

/ (Note: If priority-based interrupt routing is not implemented, an interrupt request is routed to the specified
/ target processor when the interrupt request is both pending and enabled. A single interrupt mask bit per
/ interrupt request per processor would support this simplified implementation.)

/ The Interrupt Request Registers (IRR -- one per processor) are used to identify which interrupt(s) are
/ pending for each processor. The IRR can indicate either all pending interrupts for that processor or just the
/ highest-priority pending interrupt for that processor. Software can poll the IRRs to see which interrupts are
/ pending for which processors.

/ The Interrupt Source Registers (ISR -- one per processor) are used to identify the last interrupt(s) routed to
/ that processor. The ISR can include the interrupt vector(s), the interrupt level(s) awaiting service, or other,
/ model-dependent information. If software polls for interrupts, the ISRs may not be required.

| Normally, the highest-priority pending interrupt in the IRR is transferred to the ISR when the software
| interrupt handler issues an interrupt acknowledge request. (Interrupt acknowledge is decoded by the PCI
| host bridge.) The corresponding interrupt vector is returned to the processor that issued the interrupt
| acknowledge request. If no interrupt(s) are pending, interrupt acknowledge returns a model-dependent spu-
| rious interrupt vector, typically all zeros or all ones.

/ The Available Processor Mask (APM) is used to ensure that only those processors currently in the config-
/ uration participate in interrupt routing.

/ Interrupt prioritization and routing can be implemented many ways. One way is for hardware to periodically
/ cycle among all pending interrupt requests. The highest-priority pending interrupt(s) for each processor are
/ stored in that processor's interrupt request register. Interrupts which can be routed to more than one
/ processor are stored in the IRR corresponding to the processor with the lowest CPPR. If the highest-
/ priority pending interrupt in a processor's IRR has a higher priority than that processor's CPPR, the inter-
/ rupt (INT) line to that processor is asserted.

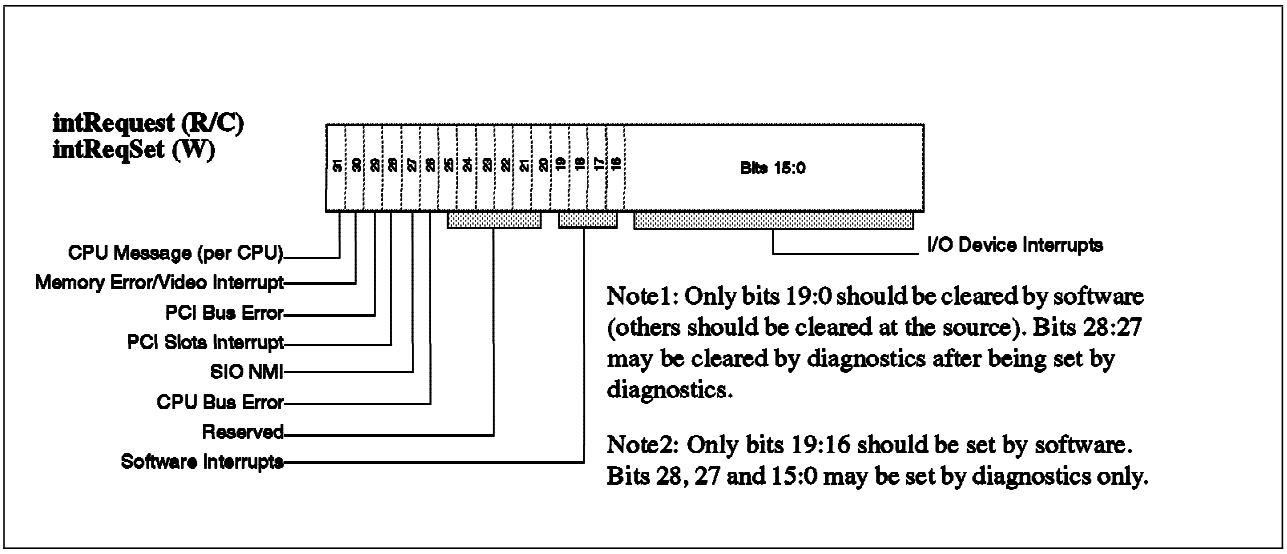
/ All the registers in Figure 52 can be read and written by software. All registers should be read and/or
/ written 32 bits at a time to minimize Bi-Endian design issues. Unimplemented bits return zeros on a read.
/ Reads, except for interrupt acknowledge requests, have no side effects.

/ A.6.5 MP Interrupt Controller Example B

/ This is an example of a lower-cost MP interrupt controller. This design meets all the requirements for MP
/ interrupts listed in Section 3.13, "Multiprocessor Considerations" but lacks some of the features of the pre-
/ vious example, most notably priority-based interrupt routing.

/ A.6.5.1 Interrupt Logic

/ The interrupt control logic must signal to the appropriate processor interrupts from I/O devices and the
/ memory subsystem, as well as software-initiated requests. An active interrupt source that is unmasked by a
/ processor will result in a pending interrupt for that processor. That pending interrupt asserts the appropriate
/ processor's interrupt line. That processor will then service the outstanding interrupt(s).



/ Figure 53. Global Registers (One per System)

/ Features of the implementation include the following:

/ • Capacity for up to 32 interrupt sources.
/ • Independent processor (per-processor) interrupt masks.
/ • Ability to let software prioritize interrupts by providing per-source masking as well as access to a register containing all unmasked interrupts.
/ • Per-processor message interrupts.
/ • Support for standard I/O device interrupts (software configurable).
/ • Support for four software interrupts.

/ A.6.5.2 Interrupt Handler

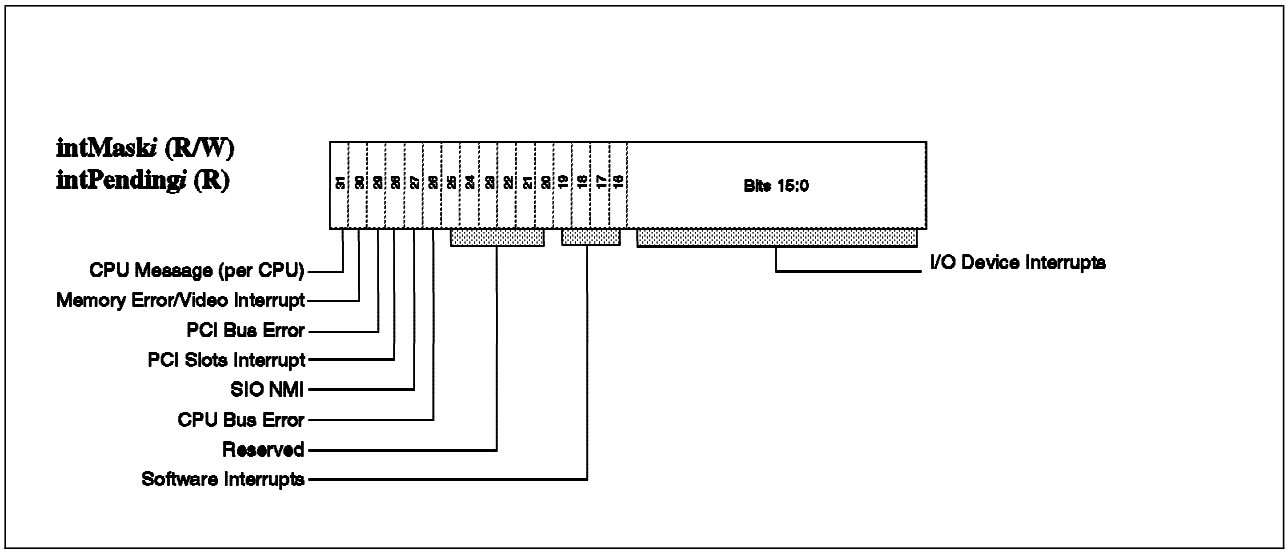
/ The software model for the interrupt handler relies on a set of software-accessible system registers. These registers are intRequest, intReqSet, intMaski, and intPendingi.

/ The implementation of the interrupt logic consists of a set of registers which define the source of each interrupt, a combinational logic per-processor interrupt generator and logic to handle reads and writes of the interrupt registers.

/ The system supports up to 32 interrupt sources. Figure 53 and Figure 54 show all possible sources for interrupts, as well as bit locations of each source. Each source has an associated bit in the intRequest register. Each of these interrupt sources has a level of masking associated with it at a per-processor level (called intMaski, where “i” is the cpu number, or cpuID). Interrupts which are unmasked propagate to the Interrupt Pending register associated with each processor (called intPendingi, where “i” is the cpuID). Whenever an Interrupt Pending register has a bit set, an interrupt is signaled to that processor. An example of this architecture is given in Figure 55.

/ Interrupt mask management in the per-processor Interrupt Mask Register (intMaski) is the responsibility of software. A suggested use is to mask specific interrupt sources to a specific processor (i.e. Memory Errors are only handled by processor 2).

/ There are program-settable message interrupt sources which are directed only at a particular processor. The interrupt hardware will direct the per-processor interrupt sources to the appropriate processors. These can then be masked off by the per-processor intMaski masks. Per-processor interrupts are not visible in the intRequest register since this register is global (not per-processor).



/ Figure 54. Per-Processor Registers (One per Processor)

/ A.6.5.3 Interrupt Request Set/Clear

/ The interrupt hardware allows software to set as well as clear the software interrupts in the intRequest reg-
/ ister. The set function is provided by writing a “1” to the appropriate bit(s) of the intReqSet register. The
/ clear function is provided by writing a “1” to the appropriate bit(s) of the intRequest register.

/ The I/O device interrupts correspond to the SIO IRQ0-15 interrupts. For every SIO Interrupt an IntAck
/ cycle is generated on the PCI bus. The IntAck cycle will yield a 4-bit encoded packet from the SIO which
/ corresponds to the highest-priority active SIO interrupt. This packet is decoded and the corresponding I/O
/ device interrupt is set in the intRequest register. Additionally, diagnostics exclusively may set these inter-
/ rupts by writing a “1” to the appropriate bit(s) of the intReqSet register. Once set, an I/O device interrupt
/ will remain asserted until cleared by software writing a “1” to the appropriate bit(s) of the intRequest reg-
/ ister.

/ Bits 31:29,26 always reflect the interrupt state of the source, and can only be set and cleared at that source.
/ Bits 28:27 behave similar to bits 31:29,26, but can also be tested via diagnostics. Bits 28:27 may be set by
/ writing the appropriate bits in intReqSet, but then MUST be cleared by writing to intRequest. Under
/ normal conditions (non-diagnostics mode), these bits simply reflect the interrupt state of the source, and are
/ set and cleared only at that source, not by software writes to intReqSet and intRequest.

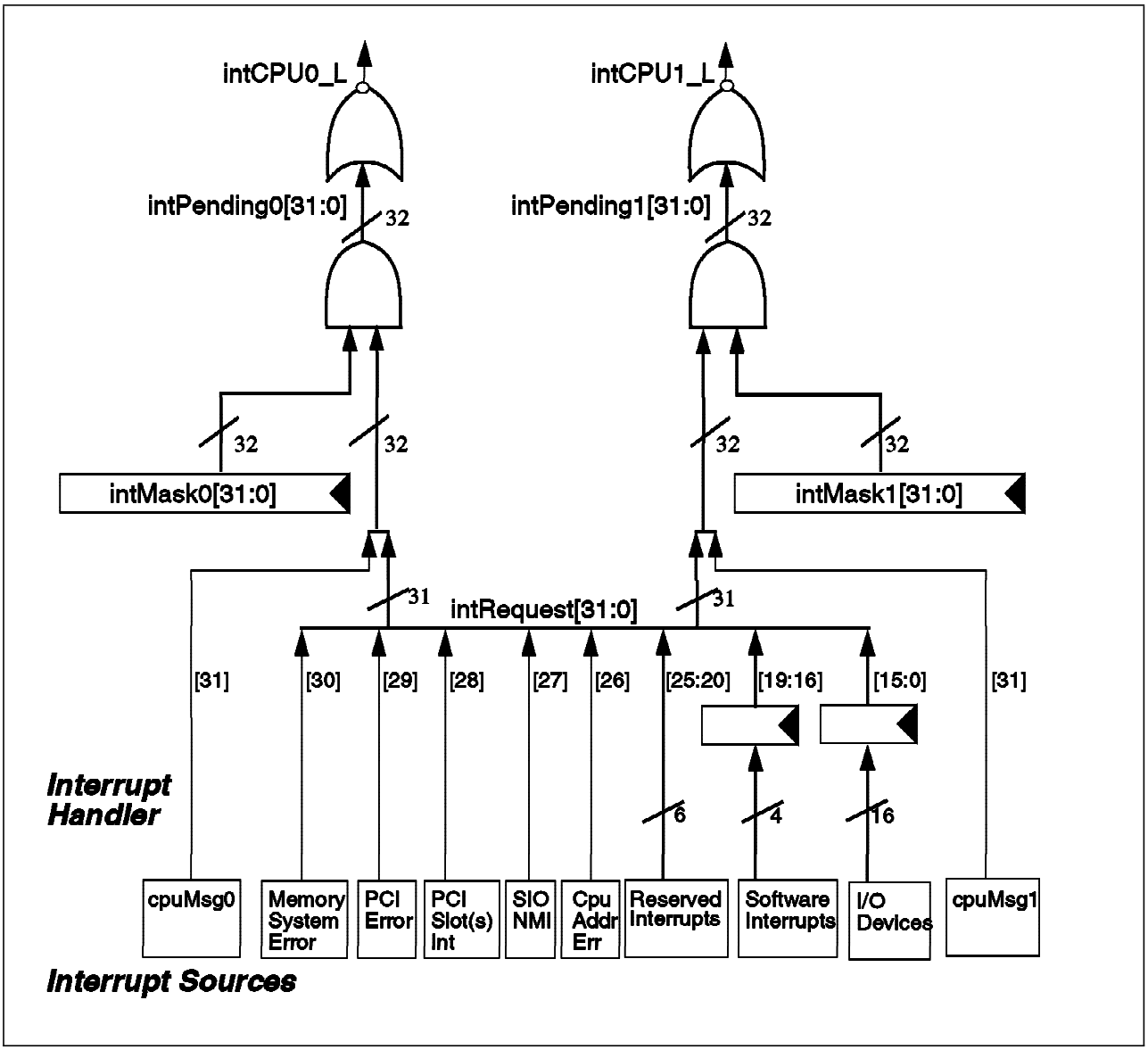
/ Bits 25:20 are reserved for future use, and should not be set or cleared.

/ A.6.5.4 Processor Interrupt Response

/ Once a processor receives an interrupt it is then software’s responsibility to read the appropriate Interrupt
/ Pending register, determine the source of the interrupt, prioritize multiple interrupts, service the highest-
/ priority interrupt at the source, and clear the corresponding interrupt request.

/ Unmasked interrupts which are reported through the intPendingi registers are updated every clock cycle.
/ This means that after an interrupt has been signalled to a CPU, new interrupts may appear in the
/ IntPendingi register. For example, it is possible for software to read the intPendingi register, mask out any
/ asserted pending interrupts (via the intMaski register) and yet still have an asserted interrupt input, but from
/ a different source requesting an interrupt since software last read intPendingi.

/ A sample interrupt handler process is as follows:



/ Figure 55. Interrupt Sources (Two-Processor Example)

/ a) Interrupted processor reads appropriate intPendingi register.
/ b) Processor branches to software interrupt handler which determines the highest-priority interrupt pending.
/ c) Source of highest-priority interrupt is pre-processed as necessary, then serviced as appropriate (e.g. device
/ driver code executed).
/ d) Device interrupt code returns to software interrupt handler for post-processing.
/ e) If source of request was bits 31:26, the request is cleared at the source; however, if source of request was
/ software or an I/O device, the bit is cleared in intRequest.
/ f) Software interrupt handler returns from interrupt.

A.7 A Proposed Diagnostic Strategy

This section describes a proposed diagnostic strategy for PowerPC Reference Platform systems. This proposal has not been adopted as a PowerPC Reference Platform architecture. It is included in this section as an example of an implementation approach which would be useful for high function systems (e.g. servers and multiprocessors). These systems would benefit from diagnoses of failing components and the resultant improved service time.

Terms used in this section include:

Error Status Registers Registers used by hardware to convey error information to firmware.
FRU Field Replaceable Unit
Fault Isolation The ability to isolate an error to a minimum number of FRUs.

Overview: When a hardware error occurs, firmware isolates the error to at most two FRUs and logs the error in NVRAM. The format of the error log in NVRAM is operating system independent. The operating system typically either reboots or stops the system after logging the error. The stand-alone diagnostics and, optionally, the operating system can subsequently be used to display the contents of the error log.

Operating systems may log additional operating system-specific error logs to NVRAM and/or to disk. These operating system-specific error logs are outside the scope of this section.

Recommendations

- It is recommended that systems implement sufficient hardware error detection and fault isolation logic to enable firmware to isolate hardware errors to at most two FRUs. Error status registers are used to convey this information to firmware.
- Time-of-failure fault isolation: When a hardware error causes a machine check interrupt, it is recommended that firmware analyze the error status registers and other information (e.g. device configuration tree) to determine:
  - which device is most likely source of the error and
  - which device reported the error.
- For PCI-related errors, it is recommended that firmware walk the device tree and read the following bits:
  - Data Parity Detected
  - Signaled System Error
  - Detected Parity Error

  in the device configuration header for the PCI host bridge(s) and the PCI devices to determine which PCI device is the most likely source of the error and which PCI device detected the error.
- While the operating system is normally involved in the processing of a machine check interrupt, it is strongly recommended that firmware not depend on an operating system to supply any of the implementation-dependent algorithms required for fault isolation.
- First and last error capture: It is recommended that firmware log each error in NVRAM. NVRAM contains a minimum of two error log entries.
- If all error log entries have been used, it is recommended that subsequent error logs overlay the most recent error log entry in a burst.

  Note: A hardware error which causes the processor to enter the checkstop state may or may not be logged in NVRAM.

- It is recommended that the system provide a way for firmware to reset the error status registers.
- It is recommended that the system provide a way for stand-alone diagnostics and operating systems to clear the error log entries in NVRAM.
- Standardized error log entry formats: It is recommended that a standard, operating system-independent error log entry format be defined for each class of errors (system board, memory, PCI, etc.).
- It is recommended that hardware systems which implement ECC memory record the failing memory address and syndrome when a correctable error occurs. It is recommended that correctable errors not

/ cause a machine check. Operating systems may periodically check for reports of correctable errors and
/ take appropriate action.

/ ***Miscellaneous***

/ • Examples of Field Replaceable Units (based on the Reference Implementation described in Section 6.0,
/ "Reference Implementation") include:
/ – System board
/ – Memory SIMM w
/ – PCI card in PCI card slot x
/ – ISA card in ISA card slot y
/ – Internal hard disk z
/ – L2 cache (and/or upgrade processor)
/ – Riser card
/ – Battery
/ – CD-ROM
/ – Flash ROM

Appendix B. Bi-Endian Design Guidance

This appendix gives design examples for Bi-Endian system implementations. These design guidelines are based on the current set of PowerPC processors (e.g. PowerPC 601, PowerPC 603, PowerPC 604). These processors assume that storage is Big-Endian. Therefore, in Little-Endian mode, a translation of data must be made somewhere in the system before the data reaches the PowerPC processor. The last sections discuss the design of a Bi-Endian graphics adaptor and the effect on these system designs if the processor architecture changes.

B.1 Little-Endian Address and Data Translation

In order for the current PowerPC processors running in Little-Endian mode to correctly access an object in Little-Endian-organized storage, the object must have its bytes show up in the correct processor byte lanes, and the Big-Endian address must be changed to refer to the correct location (after the object has had its byte lanes reordered). This translation has three parts.

The first part of the translation achieves address conversion and is done by the PowerPC processor. It is summarized below (a more detailed discussion can be found in an appendix of *The PowerPC Architecture*).

A PowerPC processor has status two bits that handle so-called Big-Endian and Little-Endian mode shifts. The Endian mode of the kernel and the Endianess of the current operating mode are recorded in two status bits (the register and bit definitions are implementation dependent and are found in PowerPC processor-specific user's manuals). Note that the PowerPC 601 uses only 1 bit. When Little-Endian mode is enabled, the effective address (EA) is modified in the PowerPC processor as shown in Table 24 before it is used to reference memory.

Table 24. Address Modification for Little-Endian Mode

Data Access	**EA Modifier**
Byte	XOR with b111
Halfword	XOR with b110
Word	XOR with b100
Doubleword	(no change)

This address modification results in correct Little-Endian addresses being presented to memory for aligned accesses (as will become clearer in the example below). The PowerPC architecture allows that unaligned accesses in Little-Endian mode trap, so incorrect memory references cannot be generated. (One subtlety here is that string operations and load and store multiple are considered unaligned accesses, and thus trap in Little-Endian mode also.)

The second part of the translation ensures that the bytes of the object addressed show up on the correct byte lanes of the processor. In Little-Endian mode, this translation requires that the bytes of a doubleword be reversed between I/O storage and the processor. This byte reversal may either happen as the data is read from or written to a Little-Endian I/O device and put into System Memory or it may occur as the data in Little-Endian order in System Memory is brought into the processor. The required byte alignment as a function of Endian mode and access is summarized in Table 25. The table assumes that the value X'1112131415161718' is stored at address 0 (actually any doubleword-aligned address.)

Table 25. Bytes Accessed Versus Endian Mode

	Byte Address								
Big-Endian	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Little-Endian**
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Byte at addr 0	11								Byte at addr 7
Byte at addr 1		12							Byte at addr 6
Byte at addr 2			13						Byte at addr 5
Byte at addr 3				14					Byte at addr 4
Byte at addr 4					15				Byte at addr 3
Byte at addr 5						16			Byte at addr 2
Byte at addr 6							17		Byte at addr 1
Byte at addr 7								18	Byte at addr 0
Halfword at 0	11	12							Halfword at 6
Halfword at 2			13	14					Halfword at 4
Halfword at 4					15	16			Halfword at 2
Halfword at 6							17	18	Halfword at 0
Word at addr 0	11	12	13	14					Word at addr 4
Word at addr 4					15	16	17	18	Word at addr 0
Doubleword at 0	11	12	13	14	15	16	17	18	Doubleword at 0
Instr at addr 0	i00	i01	i02	i03					Instr at addr 4
Instr at addr 4					i10	i11	i12	i13	Instr at addr 0

Thus a processor in Big-Endian mode that accesses the halfword at address 4 expects to see the value X'1516'. The most significant byte of the halfword, X'15', appears in byte lane 4 and the least significant byte, X'16', in byte lane 5. The table shows that applying the address mapping of Table 24 to the address and reversing the bytes between storage and the processor will result in referencing the quantity X'1516' in byte lanes 3 and 2 at halfword address 2 in a Little-Endian storage system, as required.

The third part of the translation requires that, in Little-Endian mode, addresses be generated correctly when addressing data in Little-Endian storage or when addressing I/O device addresses. For instance, since the effective address generated by the Little-Endian program as modified by the processor is used to access I/O, the programmer writing a device driver would have to precompensate the effective address used to access an adapter so that after the modification by the processor, the real address used to access the adapter is the real address of the target storage/register on the adapter. This translation must be done in every device driver and makes writing device drivers very error prone. A better solution is to solve the address modification problem one time in hardware. Logic must be added to the I/O interface such that when Little-Endian mode is selected, the added logic undoes the modification to the three low-order bits of the address. Then the unmodified (remodified) address used to access the I/O adapter is the same address as generated by the program. This system design is preferred, since a programmer writing a device driver is able to use the control register addresses as specified in the adapter hardware reference manual.

In summary, in a PowerPC Reference Platform-compliant machine, whenever the machine is running in an Endian mode different than the native mode of the processor (e.g. the PowerPC processor expects Big-Endian storage order, but the system is running in a Little-Endian mode) the address must be remapped and the byte lanes reversed somewhere on the way to or from the I/O subsystems.

B.2 Conforming Bi-Endian Designs

Several designs for implementing a Bi-Endian architecture are possible. The next two sections describe an approach in which both the memory and I/O subsystems are Bi-Endian, and a second approach in which memory is Big-Endian and the I/O subsystems are Bi-Endian.

B.2.1 Bi-Endian Memory and Bi-Endian Transportable I/O Design

The Bi-Endian Memory and Bi-Endian Transportable I/O design (e.g. the Reference Implementation) for a Bi-Endian system is shown in Figure 56. I/O devices are divided into two classes: "Transportable I/O," which consists of devices such as disks, tapes, and networks; and "Presentation I/O," which consists of devices such as graphics, audio, and video adaptors. For this design, memory, the processor and Transportable I/O interfaces must support both Endian modes. Data may exist on the Transportable devices in either Big-Endian or Little-Endian format; it is brought in, and sent out to the outside world, in either form. Presentation I/O are by design either Big-Endian or Little-Endian (or with extra hardware they may be Bi-Endian). The processor accesses both storage and I/O through a controlled byte reversal multiplexor and controlled address modification function (e.g. Xpose_bytes and Mod in Figure 56). The address modification algorithm is shown in Table 24. The byte reversal multiplexor reverses the position of each byte in a doubleword as shown in Table 26. Similarly, when the transfer between the processor and I/O is on a 4-byte I/O bus, the byte reversal is performed as shown in Table 27.

Table 26. Endian Mode Data Byte Reversal

I/O	**Storage Byte**	
Byte	**BE Mode**	**LE Mode**
0	0	7
1	1	6
2	2	5
3	3	4
4	4	3
5	5	2
6	6	1
7	7	0

Legend:

BE Mode Big-Endian Mode
LE Mode Little-Endian Mode

Table 27. Byte Reversal, Unequal Bus Widths				
I/O		**Storage Byte**		**Description**
Word	**Byte**	**BE Mode**	**LE Mode**	
0	0	0	7	Word 0: Even addressed word
	1	1	6	
	2	2	5	
	3	3	4	
1	0	4	3	Word 1: Odd addressed word
	1	5	2	
	2	6	1	
	3	7	0	

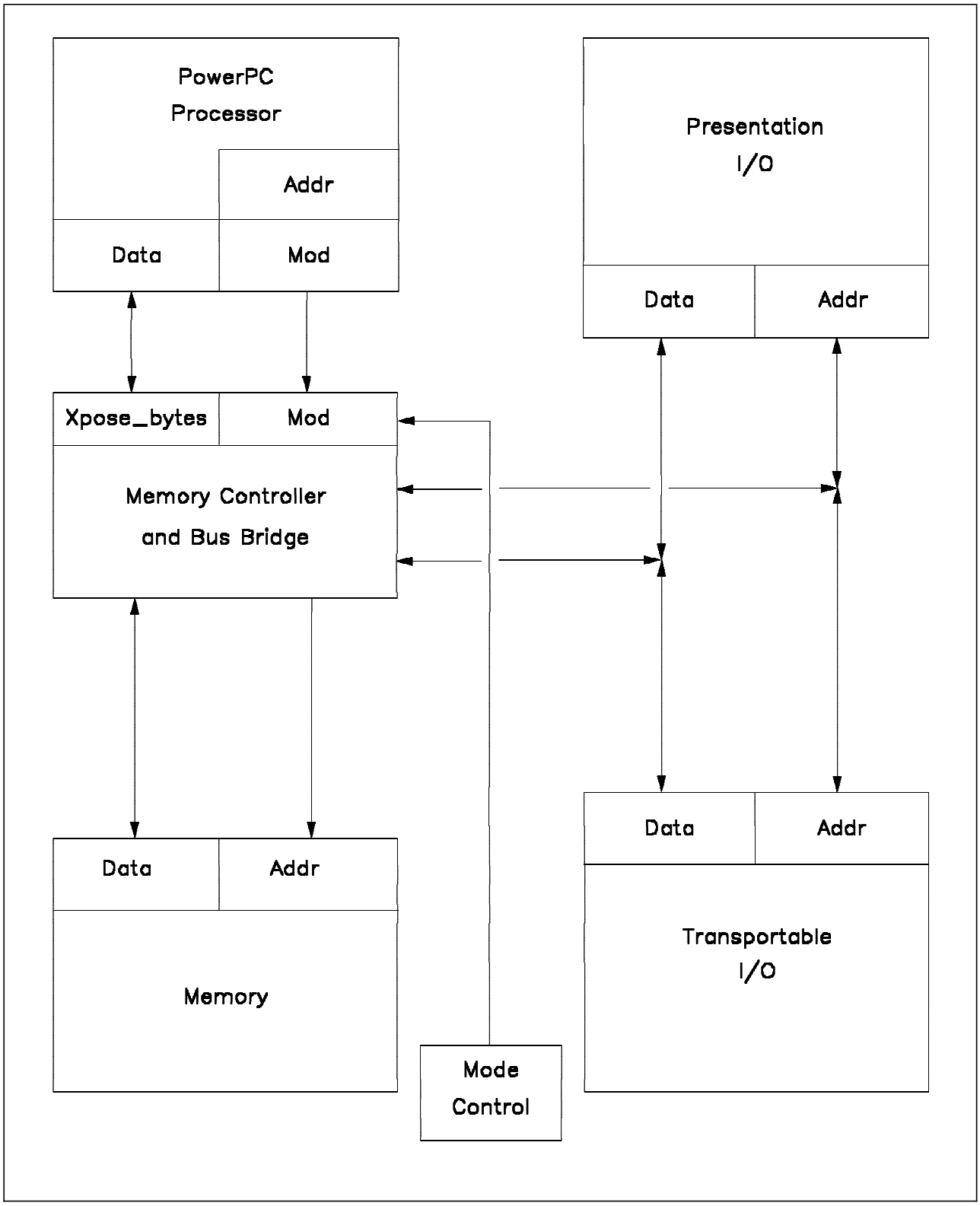


Figure 56. Bi-Endian System with Bi-Endian Memory and I/O

In this implementation, the memory and I/O are connected as if they were Big-Endian. In this case, byte 0 of storage or I/O is considered the most significant byte and passes through the controlled byte reversal multiplexor to byte 0 of the processor (MSB of the processor). The rules for applying the byte reversal multiplexor and the address modification are as follows:

Mode	Function
Big-Endian	No byte reversal of data and no address translation
Little-Endian	Byte reversal of data and address translation

From the viewpoint of Endianess of the data, I/O interfaces access only storage, not the processor or other I/O interfaces. I/O master transfers that move data from one device to another do not enter into the Endianess translation. With this in mind, areas that must be described to perform this Bi-Endian support include:

a) Processor and I/O mode control
b) Processor-to-memory interface
c) I/O-to-memory interface
d) Processor-to-I/O interface

The sections below describe these four areas. For the three interfaces, both address processing and data handling are described.

B.2.1.1 Processor and I/O Mode Control

The mode of the system may be changed by software, which has to perform the required functions to place the processor and I/O system in that mode. This design is based on the assumption that the processor and I/O are in the same Endian mode. No attempt has been made to architect a system where Little-Endian data is read and translated for a Big-Endian-mode processor running Big-Endian applications. The processor comes up in Big-Endian mode at power on or after a hardware reset. A definition in each PowerPC processor user's manual describes the process (sequence of instructions) required to change the Endian mode of that processor.

Since the processor does not provide an external signal indicating the Endian mode selected, the system must provide a mechanism to allow software to control the Endian mode of other subsystems. During configuration, software will use this mechanism to select the Endian mode to be used by storing the appropriate control value to the address of the control mechanism. The system design may place the control mechanism address in the real memory space or in the I/O Memory space, whichever is more convenient. Software must be able to address and alter the mode control mechanism in both Endian modes.

B.2.1.2 Processor-to-Memory Interface

In this form of the design, memory is assumed to be in the same Endian mode as the processor. Memory is accessed in two modes: as a result of cache-inhibited loads and stores or as a result of cache reads or writes. Each of these operations will be described below.

Operations between cache and memory are always doublewords or larger and as such the address in either Endian mode is unaffected. Data is byte reversed for Little-Endian mode and not byte reversed for Big-Endian mode. The result of these operations is that data brought into cache from Big-Endian memory is unchanged, and data brought into cache from memory in Little-Endian mode is byte reversed and stored as the PowerPC processor expects to see it (e.g. most significant byte first). Writing data back to memory from cache in Little-Endian mode reverses the byte order and restores the data in Little-Endian order.

Cache-inhibited loads and stores have to adjust the data and addresses for the expected processor formats of data and addresses. For Little-Endian mode, the address generated by an instruction points to where the data exists in Little-Endian storage. The address is translated by the processor, and then translated again outside the processor which returns it to its original value. The data is byte reversed, putting it in the order expected by the processor logic on fetches or expected in storage for stores. For Big-Endian mode the address is unchanged and the data is placed in memory in the order contained in the processor. The Refer-

ence Implementation approach would reverse the bytes once due to the connection to memory and then reverse them again in the multiplexor, restoring them to their original Big-Endian order.

B.2.1.3 I/O-to-Memory Interface

There are no address or data transformations between memory and I/O. Data is placed in memory in the same order as it exists on the I/O devices independent of Endian mode. The one exception might be Presentation I/O devices. Adaptors for these devices could be designed to accept data from Little-Endian and Big-Endian storage or to always accept data only in one format. In this case, the software would have to handle the data transformation. For example, a graphics adaptor that expected data in Little-Endian format would need software to byte reverse the data in Big-Endian mode before putting the data in memory.

B.2.1.4 Processor-to-I/O Interface

In Little-Endian mode, the address as modified by the processor must be modified again by the I/O interface such that the address used for the I/O access is the address computed by the storage instruction. Within the processor, the I/O addresses computed by a storage instruction are modified by the processor before the access is performed. Regardless of whether the access is to I/O space memory or a device control register, the address originally computed by the instruction is the address that must be used to access I/O space. The address modification algorithm shown in Table 24 is used to remodify this I/O address. This function is shown in Figure 56 in the box labeled “Mod,” which is controlled by the box labeled “Mode Control.”

The data transfer between the processor and I/O is managed in the same manner as the transfer between the processor and memory. Bytes are crossed between source and destination byte channels as indicated in Table 26 and Table 27. This byte transposition will occur once in Little-Endian mode to transform the processor-held data in Big-Endian format to Little-Endian format for I/O. In Big-Endian mode the byte transformation order is not transformed.

B.2.2 Bi-Endian I/O Design

The Bi-Endian I/O design for a Bi-Endian system is shown in Figure 57. For this design, storage is always Big-Endian, which means data is always stored with the MSB first. Transportable I/O interfaces must support both Endian modes. Data may exist on the Transportable devices in either Big-Endian or Little-Endian format; it is brought in, and sent out to the outside world, in either form. Presentation I/O are by design either Big-Endian or Little-Endian, and need not change modes since they deal with a constant visual and audio external world.

The processor accesses both storage and I/O. From the viewpoint of Endianess of the data, I/O interfaces access only storage, not the processor or other I/O interfaces. I/O master transfers that move data from one device to another do not enter into the Endianess translation. From a system viewpoint, I/O interfaces access only storage, not the processor or other I/O interfaces. Areas that must be designed to perform this Bi-Endian support include:

a) Processor and I/O mode control
b) Processor-to-memory interface
c) I/O-to-memory interface
d) Processor-to-I/O interface

The sections below describe these four areas. For the three interfaces, both address processing and data handling are described.

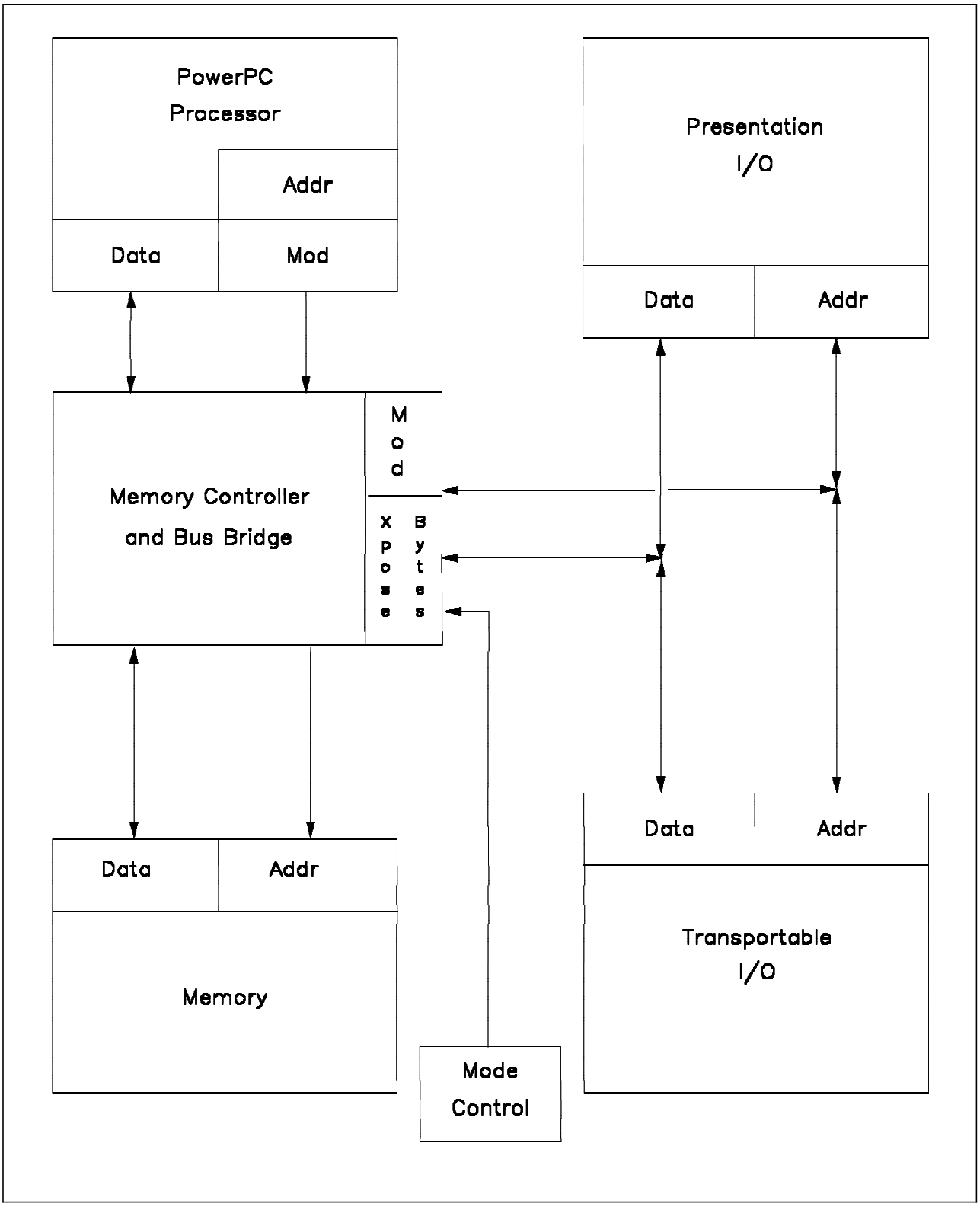


Figure 57. Bi-Endian System with Bi-Endian I/O

B.2.2.1 Processor and I/O Mode Control

The mode of the system may be changed by software, which has to perform the required functions to place the processor and I/O system in that mode. This design is based on the assumption that the processor and I/O are in the same Endian mode. No attempt has been made to architect a system where Little-Endian data is read and translated for a Big-Endian-mode processor running Big-Endian applications. The processor comes up in Big-Endian mode at power on or after a hardware reset. The definition in each PowerPC processor user's manual describes the process (sequence of instructions) required to change the Endian mode of that processor.

Since the processor does not provide an external signal indicating the Endian mode selected, the system must provide a mechanism to allow software to control the Endian mode of other subsystems. During configuration, software will use this mechanism to select the Endian mode to be used by storing the appropriate control value to the address of the control mechanism. The system design may place the control mechanism address in the real memory space or in the I/O Memory space, whichever is more convenient. Software must be able to address and alter the mode control mechanism in both Endian modes.

B.2.2.2 Processor-to-Memory Interface

Address translations are performed by the processor. Cache block transfers between the processor and storage are always a doubleword or larger and the address is not affected by the Endian mode. With Big-Endian storage, Little-Endian-mode loads and stores of aligned scalars with caching enabled work correctly after the address translation. When Little-Endian mode is enabled, loads and stores with caching inhibited use the address as modified by the processor. These instructions work the same and have the same constraints as for loads and stores out of cache.

For the processor-to- or from-memory interface, the data handling is independent of Endian mode. The memory stores multi-byte scalars in Big-Endian order, which has the most significant byte at the first byte of the address.

B.2.2.3 I/O-to-Memory Interface

In Little-Endian mode, storage addresses generated by I/O devices are modified using the address modification described in Table 24 prior to performing the access. This address modification is shown in the block labeled "Mod" on the right side of the Memory Controller and Bus Bridge. This address modification is performed for both Transportable I/O and Presentation I/O adaptors, as shown in Figure 57. This address modification is required to adjust for the format of the data in memory that has been byte reversed to place the MSB first.

Data transferred to and from the Transportable I/O devices that must switch Endian modes (e.g. disks, diskettes, communications) must have data that is stored in Little-Endian format converted to Big-Endian format. I/O transfers may be done at any implementation-determined width, but this byte reversal of data in Little-Endian mode is done by treating the data as a string of bytes that must be reversed within a doubleword (see Table 26). For unaligned transfers or transfers of less than a doubleword, bytes must be crossed from the source byte channel to the destination byte channel as shown in this table. This design places a byte-reversing multiplexor in the path from Transportable I/O to memory. This byte reversal multiplexor is shown as the "Xpose bytes" box in Figure 57.

When the interface between main storage and I/O requires a conversion from a two-word bus to a one-word bus, the byte reversal should be done in accordance with Table 27, which assumes a two-word bus to main storage and a one-word bus to I/O.

An unaligned access (e.g. read or write of System Memory) that crosses a doubleword boundary must be performed as multiple accesses. The address modification algorithm described above is not applicable to unaligned accesses. One approach to handling unaligned accesses is to perform the access as multiple aligned

accesses using byte, halfword, and word operations for which the main storage address is modified as described above.

The specific design of the Presentation I/O device adaptors will determine if a second byte-reversing multiplexor is present on the device and will influence how software device drivers must interface with this device. For instance, an audio adaptor that has byte reversal logic in it may be placed on a bus, while a graphics adaptor that does not have byte reversal logic may be on the same or a different bus. Depending upon the mode of the processor and memory, neither device may need byte reversal or both may need it. For example, a graphics adaptor with a Little-Endian design (e.g. registers and data are expected in Little-Endian order) could be addressed by a Big-Endian processor and by storage, via software performing the byte reversal, before the Big-Endian data was sent to the graphics adaptor. Alternatively, the graphics adaptor could be designed to perform the byte reversal. In this case, the adaptor would pass data straight through when the processor is in Little-Endian mode and do a byte reversal when the processor is in Big-Endian mode.

B.2.2.4 Processor-to-I/O Interface

In Little-Endian mode, the address as modified by the processor must be modified again by the I/O interface such that the address used for the I/O access is the address computed by the storage instruction. Within the processor, the I/O addresses computed by a storage instruction are modified by the processor before the access is performed. Regardless of whether the access is to I/O space memory or a device control register, the address originally computed by the instruction is the address that must be used to access I/O space. The address modification algorithm shown in Table 24 is used to remodify this I/O address. This function is shown in Figure 57 in the boxes labeled “Mod.”

The data transfer between the processor and I/O is managed in the same manner as the transfer between I/O and memory except that the processor is the master (it provides address and control) rather than an I/O mechanism. Bytes are crossed between source and destination byte channels as indicated in Table 26 and Table 27. This byte reversal is performed on all I/O between transportable devices and may be performed on I/O to presentation devices. The processor performs unaligned accesses as multiple accesses to aligned doublewords and may transfer an odd number of bytes within an aligned doubleword.

B.3 Software Support for Bi-Endian Operation

The Endian mode-switching logic should be a software abstraction provided by each operating system. The specific set of instructions are processor dependent.

As pointed out above, data for Presentation I/O and control data for any I/O device may have to be byte reversed. Services to perform these operations should be provided by the support software. Typically this would be language syntax and compiler support for a load and store with byte reversal of 2-, 4- and 8-byte scalars. If the compilers for all languages do not support these forms of loads and stores, then the operating system should supply services that perform the byte reversal.

B.4 Bi-Modal Devices

For performance reasons, applications in many operating environments write directly to graphics adaptors. It is recommended that graphics adaptors used in PowerPC Reference Platform systems provide both Big-Endian and Little-Endian data transfer methods. As of this document publication date, AIX requires a Big-Endian graphics adaptor interface, while all other operating systems that support the PowerPC Reference Platform require a Little-Endian graphics adaptor interface. It is recommended that graphics subsystems implementing bi-modal support use the design shown in Figure 58, which applies to both the frame buffer and the graphics subsystem’s register/command space. This is necessary in order to provide for adequate per-

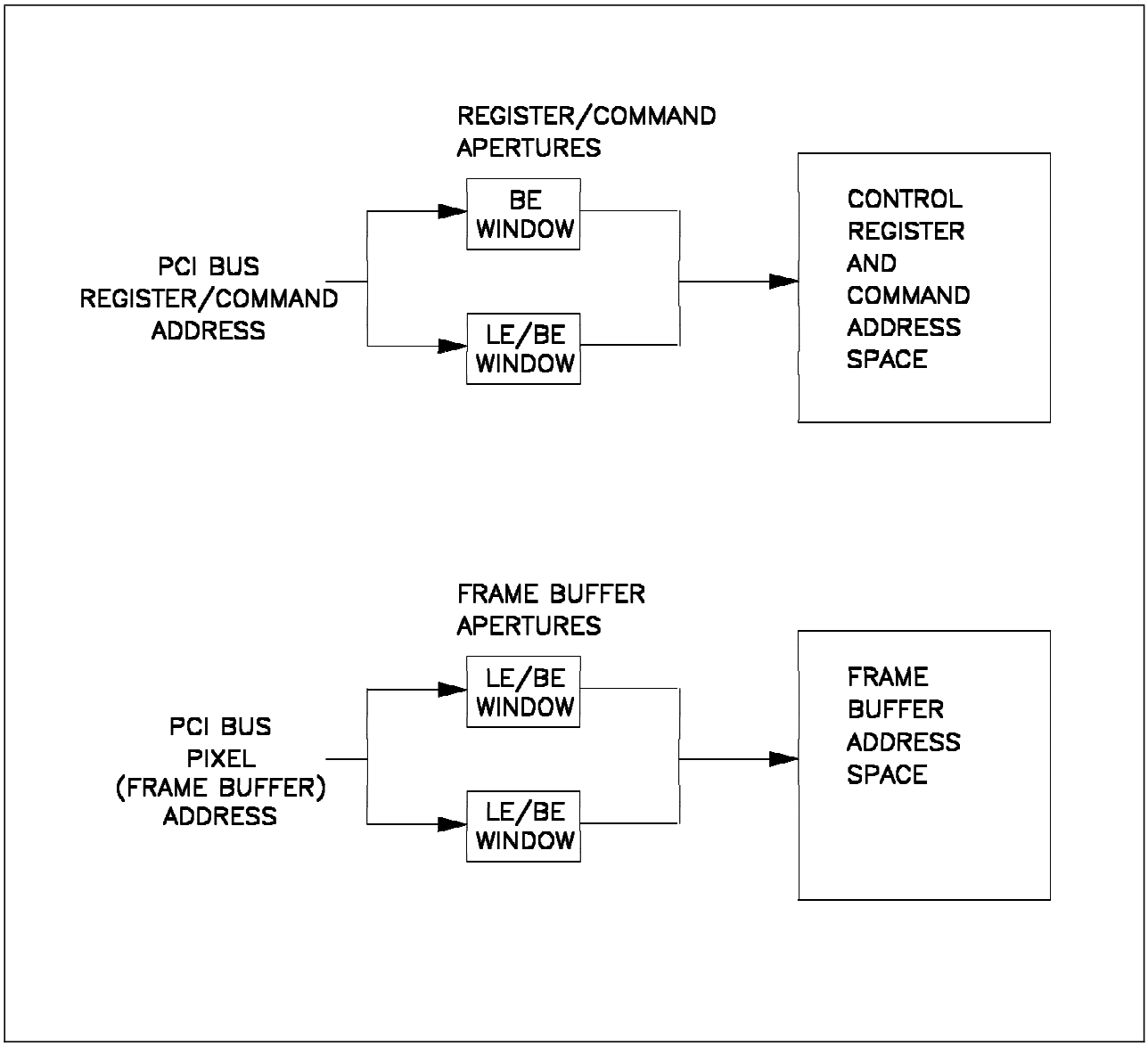


Figure 58. Bi-Endian Apertures for the Graphics Subsystem

formance when using the hardware acceleration features of the graphics subsystem. In this design, the graphics subsystem is accessed through address “apertures” which perform the Endian switch of data as it passes through the aperture.

The register/command apertures provide two ways (Big- or Little-Endian) to access the control data. They are defined address ranges within the address map of the adaptor. While two dedicated apertures are desired, one aperture may be used if it can programmably accommodate Big- or Little-Endian. The aperture labelled “BE” would always provide byte swapping in a two-aperture implementation. The aperture labelled “LE/BE” would provide no byte swapping in a two-aperture implementation, and provide byte swapping in a single-aperture implementation based on one of the following:

a) The size of the facility being accessed would automatically determine how swapping is to occur for a particular access (e.g. half- or full-word, or no swapping).
b) Software would programmably set the swapping function to either halfword, full-word, or none as appropriate.

The frame buffer apertures provide two ways to access the pixel data. The frame buffer apertures are defined address ranges within the address map of the adaptor. Each frame buffer aperture can be independently controlled to provide one of the following modes, under software control:

a) No swapping
b) Byte swapping within each halfword, e.g. A-B-C-D = > B-A-D-C
   - This becomes A-B-C-D-E-F-G-H = > B-A-D-C-F-E-H-G for a system with a 64-bit data interface, such as the PCI bus with optional 64-bit bus extension
c) Byte swapping across the entire word, e.g. A-B-C-D = > D-C-B-A
   - This becomes A-B-C-D-E-F-G-H = > D-C-B-A-H-G-F-E for a system with a 64-bit data interface, such as the PCI bus with optional 64-bit bus extension
d) Byte swapping across a doubleword, e.g. A-B-C-D-E-F-G-H = > H-G-F-E-D-C-B-A, for 64-bit pixels on the 64-bit extended data bus

Note: Determination of the correct swapping mode for a given access may be performed automatically in the hardware. This approach is not recommended if there is a possibility that the pixel depth in the frame buffer can be interpreted differently by multiple devices, or is not guaranteed to be “known” or implied by the state of the graphics subsystem hardware.

With this capability, one aperture could access the frame buffer in Little-Endian mode, while the other could access it in one of the Big-Endian modes. Similarly, one aperture could be defined to swap for 16-bit pixels, / while the other could be defined to swap for 24- or 32-bit pixels. Alternatively, several apertures may be / defined to support the various pixel depths.

Systems employing only one frame buffer aperture would provide the previously described swapping options under software control.

Twenty-four-bit pixels are defined to occupy the least significant 3 bytes of a full word. The most significant byte may be used as an alpha byte, or may be unused. Sixteen-bit pixels are always halfword aligned, and 24- or 32-bit pixels are always full-word aligned.

Pixels between 32 and 64 bits are defined to occupy the least significant bytes in the 64-bit field, and are doubleword aligned.

B.5 Future Directions in Bi-Endian Architecture

This section discusses directions in the design of future PowerPC chips and the effect these changes will have on PowerPC Reference Platform system design. The current implementation of Little-Endian in the PowerPC chips reduces internal processor complexity by moving some of the Bi-Endian support out of the processor. This implementation has two disadvantages:

a) The PowerPC processor chip expects data presented in Big-Endian order.
b) The processor in Little-Endian mode interrupts when it encounters unaligned loads and stores, multiple loads and stores, and string loads and stores.

The first disadvantage impacts cost, complexity, and chip portability. For Little-Endian hardware implementations, the current processor requires byte reversal and address translation hardware external to the processor chip. This extra hardware adds complexity and cost to the machines that are being designed and limits the applicability of the processor chip exclusively to designs that have this logic. Without this restriction, the processor chip could be incorporated into Little-Endian designs without modification to that design.

The second disadvantage is a performance and portability impact. Performance is impacted because if a program uses unaligned loads and stores, load or store multiple instructions, or string operations, then the interrupt handler must emulate these instructions. For proper operation, the interrupt handler must translate

unaligned loads and stores to loads and stores of aligned pieces, and must translate load and store multiple instructions and string operations to loops. The Little-Endian program has a performance impact from the interrupt and the emulation. Alternatively, the Little-Endian source code which is to be transported to a PowerPC environment may be changed to eliminate unaligned mappings, and compilers may be modified to not generate multiple loads and stores or string operations. This requirement impacts portability of application source code and requires different logic in compilers supporting Little-Endian-mode PowerPC implementation.

Future versions of the architecture may permit PowerPC processor chips to support a true implementation of the Little-Endian mode. In this true Little-Endian mode, data would go to the processor in Little-Endian format and be addressed from the processor with the Little-Endian address. Additional hardware support for Little-Endian unaligned operations might be provided.

Figure 59 shows the design to be used with any future version of the PowerPC processor that implements full Bi-Endian support. No address modifications or byte reversal multiplexors are required. The previous designs may be migrated to this design by physically removing the byte reversal multiplexors and address modification logic components from the design, or by functionally removing them by changing the rules under which they are applied. For instance, the Reference Implementation of the Bi-Endian Memory and Bi-Endian Transportable I/O design, Figure 56, would require the address modification and byte-transposing multiplexor to always be off. The address modification is no longer needed, because the processor would not modify the address since it deals with Little-Endian data in Little-Endian order. The byte reversal is not required because the processor accepts data in Little-Endian order.

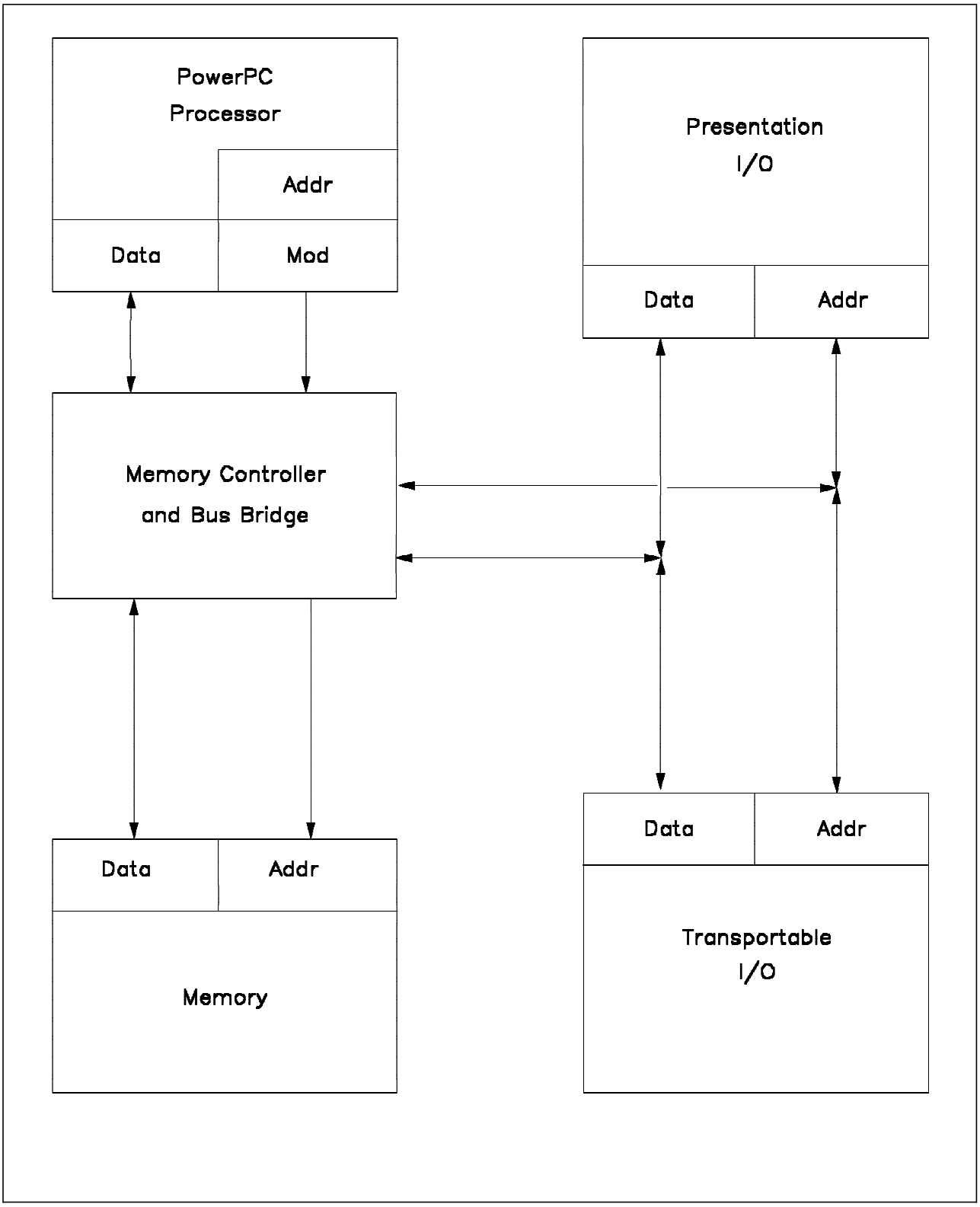


Figure 59. Design with a Full Bi-Endian Processor

Appendix C. Additional Compliant Subsystems and Devices

/ This section lists devices which can be used to configure a PowerPC Reference Implementation. As of the
/ publication date of this specification, this list of adaptors supports this PowerPC Reference Implementation.
/ As additional devices gain software support, they will be added to this list. The last subsection lists specific
/ regional device characteristics which should be considered.

/ C.1 Native Subsystems

/ It is strongly recommended that all operating systems support the items in the following list of directly
/ attached adaptors and devices. For maximum compatibility, system designers should use these devices. The
/ adaptors and devices listed are identical to the same adaptors and devices used in Intel-based PCs. It is
/ expected that the majority of these devices have Intel-based device drivers, simplifying the port to PowerPC
/ Reference Platform systems.

- Keyboards
  - / – Space-saving 84-, 85- and 89-key
  - | – Enhanced keyboard
- Pointing Devices
  - – PS/2 Mouse and compatible
  - | – TrackPoint II and TrackPoint III
- SCSI Adaptors
  - – NCR810/720 (clones may require device drivers for Adaptec and Future Domain adaptors)
- SCSI-Attached Devices:
  - – Hardfiles
    - — All hardfiles compatible with standard SCSI interface; sizes range from 240 MB to 2 GB.
  - – CD-ROMs
    - — Toshiba 4401/3401 and compatible
  - – Tapes
    - — IBM 1.2 GB External
    - — 2.0 GB HP 35470A DAT
    - — HP 35480 DAT
    - — WangDat 3200
    - — Wangtek 5525ES (QIC)
    - — 51000ES (QIC)
    - — Tandberg 3820
    - — Tandberg 4120
  - – Miscellaneous
    - — IBM 3.5" optical
    - — IBM R/W optical
    - — Microarray RAID5 Disk Array
    - — OASIS RAID5 Disk Array
  - – Scanners
    - — HP ScanJet IIp/c and compatible (VEGA standard)
- / IDE
  - / – Opti 82C621 Enhanced IDE

- Floppy Adaptor
  - Intel 82077 and Compatible
- Floppy Drives
  - Non-IBM 720 KB/1.44 KB/2.88 KB 3.5" and 1.2 MB 5.25"
- Audio Adaptor
  - AD1848 (Business Audio Compatible)
  - Crystal Semiconductor CS4231
- Serial Ports
  - 16550 and Compatible
- Parallel Ports
  - Standard Parallel Port, Enhanced Parallel Port, Extended Capabilities Port
- Printers (Only vendors are listed. Epson (dot), HP (PCL), HP (plotter), and Postscript-compatible printers are supported.)
  - Brother
  - Canon
  - Citizen
  - Epson
  - Fujitsu
  - HP
  - IBM
  - Kyocera
  - Lexmark
  - OKI
  - Panasonic
  - QMS
  - Tektronix

C.2 PCI Subsystems

It is strongly recommended that all operating systems support the items in the following list of PCI adaptors and devices. For maximum compatibility, system designers should use these devices. The adaptors and devices listed are identical to the same adaptors and devices used in Intel-based PCs. It is expected that the majority of these devices have Intel-based device drivers, simplifying the port to the PowerPC Reference Platform system.

- Graphics Controllers (1024x768x8, 1024x768x16, 1280x1024x8)
  - S3 928 (used in development) or newer Vision864
  - Weitek 9000 (used in development) or newer 9100 (Bi-Endian capable)
  - IBM PowerGTX 150P (Bi-Endian capable)
  - Western Digital 90C24A2 as used in the Portable

C.3 PCMCIA Subsystems

It is strongly recommended that all operating systems support the PCMCIA adaptors listed below. For maximum compatibility, system designers should use these devices. These PCMCIA adaptors are identical to the ones used in Intel-based PC systems. The adaptors are also similar to their ISA-based counterparts. The main support required is PCMCIA socket services. The majority of the following adaptors are for communications.

- Ethernet
  - IBM PCMCIA Ethernet Card
  - Xircom PCMCIA Ethernet Card
- Modems (Data/Fax)
  - AT&T Paradyne 14.4/9.6 PCMCIA
  - Hayes 2.4/9.6 PCMCIA
  - IBM PCMCIA Modems
  - Megahertz 14.4/14.4 PCMCIA
  - Megahertz 2.4/9.6 PCMCIA
- Hardfiles
  - Maxtor 105-MB Type 3 PCMCIA
  - Sundisk Silicon Hardfiles
- Token Ring
  - IBM PCMCIA Token Ring Card
- Audio
  - IBM MWAVE PCMCIA Audio Cards
- Pager
  - Motorola Newscard**
- ISA-to-PCMCIA Bridge
  - Intel 82365SL
  - Ricoh RF5C266
  - Ricoh RF5C366
  - Cirrus CL_PD6710
  - Cirrus CL_PD6720
  - IBM CTLISA2 (2 ports)

C.4 ISA Subsystems

It is strongly recommended that all operating systems support the items in the following list of ISA adaptors and devices. For maximum compatibility, system designers should use these devices. The adaptors and devices listed are identical to the same adaptors and devices used in Intel-based PCs. It is expected that the majority of these devices have Intel-based device drivers, simplifying the port to the PowerPC Reference Platform system.

- Network Adaptors
  - 3Com Etherlink ISA Adaptors
  - Intel EtherExpress 16
  - Novell NE2000/3200 (ISA)

  - Ungermann-Bass NIUPS Adaptors
  - Standard Microsystems Ethercard ISA Adaptors
  - IBM Token Ring or Ethernet Adaptors

- Fax/Data Modems (most are Hayes compatible and use a serial port interface)
  - Hayes Modems (2400 - 14400) and Compatible
  - Intel SatisFAXtion Modems
  - Megahertz Modems
  - Practical Peripherals Modems
  - US Robotics Modems

- Other Adaptors
  - IBM M-Wave Products (ISA)
  - Creative Labs Sound Blaster Pro/16 (ISA)

C.5 Regional Device Characteristics

Within some countries or regions, common usage suggests that additional device characteristics be considered for PowerPC Reference Platform systems. These considerations are listed below:

Country	Suggested device characteristics
Japan	The 3.5-inch floppy drives should be capable of 1.2 MB as well as the 1.44-MB format. To support this 1.2-MB format, the drive may have to support a 360-rpm speed in addition to the 300-rpm speed.
Europe	Ergonomics standards may affect the system design.

Appendix D. Windows NT

This appendix describes the Windows NT operating system which will run on PowerPC Reference Platform-compliant machines.

D.1 Operating System Scope

/ The Windows NT operating system supports portable, desktop and server environments. The operating
| system has dynamically loadable software device drivers and a software layer that can be used for hardware
| adaptation. This Hardware Abstraction Layer (HAL) and the device drivers allow hardware vendors some
/ freedom to differentiate.

/ A Windows NT porting center is available to help vendors who are interested in porting their products to
/ this platform. The center can be contacted by telephone at 1-800-803-0110 or 1-206-889-9011, or by elec-
/ tronic mail on Internet at winntppc@vnet.ibm.com.

D.2 Operating System Version

/ The current version of Windows NT will support this implementation.

/ D.3 Operating System Environment

/ Windows NT runs in the Little-Endian mode.

D.4 Operating System Configuration

/ For the purpose of describing configurations which support Windows NT, three configurations are listed
/ below:

Configuration Name	Configuration Definition
Client Workstation	A basic single-user system which operates either stand-alone or as a client in a network. Software includes the base system, utilities, file systems, shared printers, performance/event monitoring, backup, remote access, network client support, system management and several personal productivity aids such as mail and workgroup scheduling.
Developer Workstation	The developer workstation is essentially the same as the client workstation. A developer might need additional disk space and would purchase compilers and debuggers to assist in application development.
Server	A configuration which provides shared system management such as workgroup-wide passwords and protection attributes. It also provides RAID and disk mirroring support.

D.5 Hardware Configuration Requirements

This section defines the minimum, optional and alternative hardware configuration requirements for three Windows NT operating system configurations. In many cases operating performance improvements can be realized by configuring systems with larger or faster components. Because these configuration adjustments are very application load dependent, no attempt has been made to recommend operationally tuned configurations.

The subsections below describe the configuration alternatives for each of the three configurations. This material is summarized in Table 29.

D.5.1 Processor Subsystem

/ A PowerPC Reference Platform-compliant system must have a PowerPC compliant processor. Windows
/ NT will support PowerPC 601, 603, 604 or compatible processors running at supported frequencies. Performance will be better with higher frequencies and with PowerPC processors having performance-enhancing features (i.e. larger internal cache).

D.5.2 Memory Subsystem

System Memory for the client workstation should be at least 16 MB. For the developer workstation, System Memory should be 24 MB. For a server, System Memory should be 32 MB. Performance improvements can be realized in all three configurations with additional System Memory.

Boot memory will not vary between configurations. The minimum size is determined by what a vendor needs to boot the machine into the state expected by the operating system as defined in Section 5.0, “Boot Process and Firmware.”

An additional cache external to the processor is optional on a PowerPC Reference Platform-compliant system. Minimum configurations for all three workstation configurations do not require an L2 cache. For performance reasons, the developer workstation could have a 256-KB L2 cache and the server could have a 512-KB L2 Cache.

Non-volatile Memory to support Windows NT will not vary between workstation configurations.

D.5.3 Storage Subsystems

The minimum hardfile size for a client workstation is 200 MB. A developer workstation would require at least 400 MB. The minimum size for a server depends upon its function. A simple print server with only one printer could be 400 MB. A data server would need significantly more hardfile space in the range of 2 to 4 GB.

A single 1.44-MB 3.5-inch floppy drive should be the minimum configuration for all three workstation types. Windows NT also supports the 2.88-MB 3.5-inch floppy drive format.

A CD-ROM is required in the minimum configuration for all Windows NT machines. Windows NT will only be shipped on this media. Product manuals, information, and command references are available through this media. An acceptable alternative to locally attached CD-ROM is network access to material on CD-ROM. The developer workstation and the server workstations could benefit from CD-ROMs with high access speeds.

D.5.4 Human Interface Subsystem

All configurations must have a graphics resolution of at least 640 by 480 pixels.

/ A keyboard and pointing device are required on all configurations.

A business audio device is required.

All configurations need the standard Real-Time Clock.

D.5.5 Connectivity Subsystem

/ Windows NT does not require a serial port, but a serial port is required as part of the standard hardware
/ configuration described in Section 2.0, “Hardware Configuration.” Windows NT supports EIA/TIA-232E-compliant serial ports. Any configuration may optionally add one or more serial ports.

None of the three configurations require a parallel port. Windows NT supports 8-bit bidirectional Compatibility Mode-compliant parallel ports.

No network connection is required. Both Ethernet and Token Ring are supported, as is full remote access over phone lines.

D.5.6 Expansion Bus Interfaces

/ Windows NT supports a PCI secondary bus. Additional parallel PCI buses or different secondary buses
/ could be used, but the vendor would have to supply modified abstraction software.

The ISA bus is optional for all three configurations. As I/O adaptors become readily available on the PCI bus (or alternative secondary bus), configurations may remove the ISA bus.

Windows NT will support PCMCIA, but a vendor including this connector must supply a compatible device driver.

Windows NT supports multiple SCSI interfaces.

Windows NT supports IDE access to disks.

/ D.5.7 Security and System Management

/ Windows NT does not require special features for security and system management.

D.6 Hardware Configuration Recommendations

The Windows NT operating system which supports PowerPC Reference Platform-compliant systems will support additional equipment. In some cases, drivers will have to be supplied by either the system vendor or the device vendor.

/ Tape drives are useful additions to PowerPC Reference Platform-compliant systems as backup and archive
/ devices. Windows NT supports three formats: Quarter Inch Cartridge (QIC), 4 mm, and 8 mm.

Multimedia upgrades with better sound and MIDI support would be useful additions to a system. The vendor would have to provide NT-compatible drivers.

Alternative graphics adaptors may be supplied on PowerPC Reference Platform-compliant machines.
/ Windows NT currently supports S3, Weitek, and Western Digital. Vendors would have to provide drivers for other adaptors.

D.7 Boot Time Abstraction Requirements

/ Windows NT uses the boot devices defined by the firmware. Only those devices (e.g. video, keyboard, disk)
/ known to the firmware participate in the boot process.

D.8 Hardware Abstraction Layer

/ The following table indicates the effect that changes in hardware features may have on Windows NT. The
/ left column lists those features of the hardware that may impact the operating system if changed. Some
/ features listed do not cause a system change. These are features that one would expect to cause operating
/ system changes, but in the case of NT, do not. The next four columns indicate which major operating
/ system components are affected by the feature change. *Firmware* refers to the ARC firmware that provides
| hardware-specific function to the hardware-independent OS Loader. *HAL* refers to the software layer that
| provides the hardware adaptation function. Firmware and HAL can be changed by hardware system
/ vendors and shipped with their products. Kernel changes are made only by Microsoft and Windows NT
/ source licensers such as IBM and Motorola. Device drivers can be distributed between Microsoft releases via
/ CompuServe and bulletin boards as well as with hardware.

/ An "X" in a box indicates that changing the hardware feature causes changes in the indicated operating
/ system component. An "[X]" indicates that the current beta version of the PowerPC port requires a change
/ but may not require change in the final release. The Currently Supported column lists the current parts that
/ are supported with the existing software.

/ This chart indicates the need to look for potential consequences of feature changes. The specific effects of
/ any hardware change must be evaluated to determine the extent of software modification required. For
/ instance, changing the RTC is a straightforward process, while changing the coherency protocol for the cache
/ could be a major effort.

/ Hardware system vendors who require firmware changes will need to obtain a Firmware Kit and the appro-
/ priate licenses from IBM or Motorola. A license may also be required from Microsoft. Hardware system
/ vendors will be notified of this requirement when they request the Firmware Kit. For HAL changes, hard-
/ ware system vendors must obtain a HAL license from Microsoft and a HAL kit from IBM or Motorola.
/ For device driver changes, a Device Driver Kit from IBM or Motorola is required.

/ Table 28 (Page 1 of 2). Effect of Hardware Changes on Windows NT

/ /	Component	Firmware	Kernel	HAL	Device Driver	Currently Supported
/	PROCESSOR:					601, 603, 604
/	Numbers of Processors	X	X	X		
/	Speed					
/	Cache Organization	X		X		
/	Cache Size	X		[X]		
/	SPRs	X	X	X		
/	TLB		X			

Table 28 (Page 2 of 2). Effect of Hardware Changes on Windows NT

Component	Firmware	Kernel	HAL	Device Driver	Currently Supported
Memory Management		X			
Processor clock			X		
Exception Vector	X	X	X		
L2 CACHE:					
Size	X		X		256 KB
Coherency	X	X	X		
MEMORY CONTROLLER:					IBM 27 82650
Speed					
Interrupt Acknowledge	X		X		
PCI Configuration Interface	X		X		
Memory Timing	X				
BUS BRIDGE CONTROLLER:					Intel SIO 82378IB/ZB
Speed	X				
Interrupt Controller	X		X		8259
Timer			X		8254
DMA Controller			X		
SUPER I/O	X		X		SMC FDC37C665, National PC87312
REAL-TIME CLOCK	[X]		X		Dallas 1385S, 1585S
POWER MANAGEMENT	X		X		
NVRAM	X		X		Dallas 1385S, 1585S
SCSI SUBSYSTEM	X			X	NCR 810
GRAPHICS SUBSYSTEM	X		[X]	X	S3 928 & 864, Weitek 9000 & 9100, WDC 90C24A2
KEYBOARD CONTROLLER	X			X	i8042
PERIPHERAL BUS:					PCI/ISA
Type	X		X		
Number			X		
Peripheral Devices				X	

D.9 Device Driver Model

There is a separate Device Driver Development Kit (DDK) available from both Microsoft and Motorola. It contains all the documentation and sample code needed to develop device drivers.

D.10 Multiprocessing Model

Windows NT supports SMP.

D.11 Application Model

Reference the ABI/API document for Windows NT.

D.12 Configuration Summary Table

Table 29 gives the minimum configuration information for each of the three Windows NT system configurations. This table specifies levels of hardware necessary to run the three Windows NT system configurations. In many cases upgrade options are possible. Some of those options and configuration alternatives were described in Section D.5, “Hardware Configuration Requirements.”

Table 29 (Page 1 of 2). Hardware Requirements for Windows NT System Configurations

	System Component	Client	Developer	Server
	Processor	PowerPC 601, 603, 604 or compatible processor		
	System Memory	16 MB	24 MB	32 MB
	Boot Memory	Standard	Standard	Standard
	Non-volatile RAM	Standard 4 KB	Standard 4 KB	Standard 4 KB
	L2 Cache Memory	Optional	Optional	Optional
	Hardfile	200 MB	400 MB	400 MB; 2 GB
/	Floppy	Optional	Optional	Optional
/	CD-ROM*	Required	Required	Required
/	Alphanumeric Input Device	Required	Required	Required
/	Pointing Device	Required	Required	Required
/	Audio	Standard	Standard	Standard
/	Graphics	640x480	640x480	640x480
	Real-Time Clock	Standard	Standard	Standard
/	Serial Ports	Standard	Standard	Standard
	EIA/TIA-232E	Windows NT supports this serial interface		
	EIA-422	Windows NT does not support this serial interface		
	Parallel Ports	Optional	Optional	Optional
	Compatibility Mode	Windows NT supports this parallel interface		

Table 29 (Page 2 of 2). Hardware Requirements for Windows NT System Configurations

System Component	Client	Developer	Server
Extended Capabilities Port	Windows NT does not support this Parallel interface		
Network Adaptors	Optional	Optional	Required
Ethernet	Windows NT supports Ethernet		
Token Ring	Windows NT supports Token Ring		
SCSI	Optional	Optional	Optional
IDE	Optional	Optional	Optional
PCI Bus	1 Required	1 Required	1 Required
ISA Bus	Optional	Optional	Optional
PCMCIA	Optional -- but a vendor-supplied device driver is required		
Tape Drive	Optional -- but a vendor-supplied device driver is required		

Legend:

Entry	Definition
Standard	The hardware configuration in Section 2.0, “Hardware Configuration,” requires this component.
Required	This component must be present in the system on which Windows NT will run.
Optional	This component is supported by Windows NT, but is not required.
*****	Capability can be provided via a network.

Appendix E. AIX

This appendix provides an overview of the AIX (R) Operating System which will run on PowerPC Reference Platform-compliant machines.

E.1 Operating System Scope

AIX, IBM's implementation of the UNIX** operating system, offers scalability from entry-level systems to symmetric multi-processing (SMP) servers suitable for enterprise mission-critical applications. The combination of networking capabilities, graphics, usability, performance, packaging, and standards compliance built into AIX make it an attractive operating environment in standalone configurations or as a client or server in networked environments.

AIX has dynamically loadable software device drivers and is modularized to assist in hardware adaption, allowing vendors the freedom to differentiate their system offerings.

AIX Version 4.1, announced July 26, 1994 and refreshed with AIX Version 4.1.1 on October 4, 1994, is the most significant enhancement to AIX since its initial introduction. AIX Version 4.1.1 features:

- SMP support
- Installation packages tailored for client and server configurations
- Improvements such as greater-than-2-GB file systems, AIX kernel threads, and enhancements to the install process
- Journaled File System (JFS) Support for disk fragmentation and dynamic compression/decompression
- A new graphical user interface (GUI) based on the Common Desktop Environment
- Improved standards alignment for compatibility with other open systems, including the X/Open XPG4 Base Profile, PowerOpen and Spec 1170
- Improved ease of use with the addition of a GUI for installation and automatic installation of device drivers

For AIX license and product information, contact David Hall, AIX OEM Relations, at 512-838-2088.

Software vendors interested in porting their applications to AIX on PowerPC systems can contact the AIX Power Team General Information Line at 1-800-222-2363.

E.2 Operating System Version

This appendix describes AIX Version 4.1.1.

E.3 Operating System Environment

AIX runs in the Big-Endian mode. AIX supports the XCOFF object module format as defined in the *PowerOpen ABI*.

E.4 Operating System Configuration

/ AIX supports a range of system configurations. For the purpose of defining configuration size, three are
/ listed below:

/ Configuration Name	Configuration Definition
/ **Client Workstation**	A basic single-user system which operates either stand-alone or as a client in a / network.
/ **Developer Workstation**	A client configuration plus additional utilities, compilers and debuggers to assist in / application development.
/ **Workgroup Server**	An entry-level configuration which provides shared resources such as a printer or / file storage to a local work group over a Local Area Network. This configuration / includes the developer workstation functions plus server support and networking / support.

E.5 Hardware Configuration Requirements

This section defines the minimum, optional and alternative hardware configuration requirements for the three configurations. In most cases, operating performance improvements can be realized by configuring systems with larger or faster components. Because these configuration adjustments are very application load dependent, no attempt has been made to recommend operationally tuned configurations.

The subsections below describe the configuration alternatives for each of the three configurations. This material is summarized in Table 31.

E.5.1 Processor Subsystem

/ A PowerPC Reference Platform-compliant system must have a PowerPC compliant processor. AIX will
/ support PowerPC 601, 603, 604, or compatible PowerPC processors running at any supported frequency. Performance will be better with higher frequencies and with PowerPC processors having performance-enhancing features (i.e. larger internal cache).

E.5.2 Memory Subsystem

System Memory for the client workstation should be at least 16 MB. For the developer workstation, System Memory should be 24 MB. For a LAN server, System Memory should be 32 MB. System Memory should start at address zero and should be continuously populated through the maximum amount in the configuration (e.g. no holes). Performance improvements can be realized in all three configurations with additional System Memory.

System ROM will not vary between configurations. The minimum size is determined by what a vendor needs to boot the machine into the state expected by the operating system as defined in Section 5.0, “Boot Process and Firmware.”

A cache external to the processor is optional on a PowerPC Reference Platform-compliant system. Minimum configurations for all three workstation configurations do not require an L2 cache. For performance reasons the developer workstation could have a 256-KB L2 cache and the LAN server could have a 512-KB L2 Cache.

/ Non-volatile Memory to support AIX will not vary between workstation configurations. AIX conforms to
/ the mapping of NVRAM provided in this document.

E.5.3 Storage Subsystems

The minimum hardfile size for a client workstation is 200 MB. A developer workstation would require at least 400 MB. The minimum size for a LAN server depends upon its function. A simple print server with only one printer could be 400 MB. A data server would need significantly more hardfile space in the range of 2 to 4 GB.

A single 1.44-MB 3.5-inch floppy drive should be the minimum configuration for all three workstation types. AIX also supports the 2.88-MB 3.5-inch floppy drive format.

A CD-ROM is required in the minimum configuration for all AIX machines. AIX will only be shipped on this media. Product manuals, information, and command references are available through this media. An acceptable alternative to locally attached CD-ROM is network access to the material which is transmitted on CD-ROM. This alternative requires that the system boot from a network because AIX does not support installation and maintenance from a floppy. The developer workstation and the LAN server workstations could benefit from CD-ROMs with high access speeds.

E.5.4 Human Interface Subsystem

All configurations should have an alphanumeric input device. If the LAN server does not function also as a developer or client workstation, they may use a simple console (e.g tty).

A pointing device is required on the client workstation and the developer workstation. If the LAN server does not function as one of these types of workstations in addition to being a server, then it does not need a pointing device. Normally a mouse is used as the pointing device. The minimum requirement is a two-button mouse. A three-button mouse is strongly recommended because X Windows on AIX does not
/ support a two-button mouse. Alternatively, the Track Point II or Track Point III may be used.

/ A business audio device is required as part of the standard hardware configuration described in Section 2.0,
/ “Hardware Configuration,” but is not required by AIX on any workstation. A PC-type speaker may be used for error and warning sounds.

A graphics system capable of at least 800x600 pels is necessary for the client workstation. A graphics system capable of at least 1024x768 pels is the minimum configuration recommended for a developer workstation. A LAN server may have only an ASCII character video system unless it also performs the functions of a client or developer workstation. The maximum resolution currently supported by AIX is 1280x1024 pels.
/ For the best graphics performance, a Bi-Endian graphics adaptor should be provided.

All configurations need the standard DS 1385S Real-Time Clock. Current implementations depend upon this specific clock chip.

E.5.5 Connectivity Subsystem

/ All configurations require the two EIA/TIA-232E-compliant serial ports which are natively attached to the
/ system as in the Reference Implementation. The AIX kernel debugger requires a serial port when it is oper-
/ ating. AIX is not distributed with support for EIA-422. System vendors who desire this capability will have to write a device driver to support it. Any configuration may optionally add more serial ports, but device drivers will have to be supplied by the vendor.

/ All three configurations require a parallel port which is natively attached to the system as in the Reference
/ Implementation. AIX supports 8-bit bidirectional Compatibility Mode-compliant parallel ports. AIX does

not support the Extended Capabilities Port, IEEE P1284. Vendors who want to include this interface must provide a device driver. Any configuration may optionally add more parallel ports.

/ All three configurations may have optional network interfaces of either Ethernet or Token Ring.

E.5.6 Expansion Bus Interfaces

All three configurations require one and only one PCI bus.

/ The ISA bus is required for all three configurations. As I/O adaptors become readily available on the PCI bus, configurations may remove the ISA slots. However, the ISA bus decoder is required for native I/O support for such interfaces as parallel, serial, keyboard, and mouse.

AIX does not currently support PCMCIA. A vendor including this connector must supply a compatible device driver and operating system extensions.

/ AIX supports the natively attached SCSI interface as in the Reference Implementation. One or more
/ hardfiles may be attached to this interface.

/ AIX does not support IDE disks.

/ E.5.7 Security and System Management

/ AIX requires no special features for security and system management.

E.6 Hardware Configuration Recommendations

The AIX system which supports PowerPC Reference Platform-compliant systems will support additional equipment. In some cases, drivers will have to be supplied by either the system vendor or the device vendor.

/ Tape drives are useful additions to PowerPC Reference Platform-compliant systems for backup and archive.
/ AIX supports SCSI-attached tape drives in three formats: Quarter Inch Cartridge (QIC), 4 mm, and 8 mm.

Multimedia upgrades with better sound and MIDI support would be useful additions to a system. The vendor would have to provide AIX-compatible drivers.

Alternative graphics adaptors may be supplied on PowerPC Reference Platform-compliant machines. AIX
/ currently supports S3 928 or Vision 864, Weitek 9000 or 9100, or IBM PowerGTX 150P. Hardware
/ vendors may supply alternate adaptors provided that the adaptors work on PCI buses, have low-level and X
/ Windows drivers compatible with AIX, and are supported by the boot process.

E.7 Boot Time Abstraction Requirements

Certain memory map requirements must be satisfied for AIX to be booted and to operate. Low memory from X'00000000' through X'00004FFF' must be reserved for system use. Table 30 shows the allocation of this real memory space. The AIX boot process requires that the space from this point through 2 MB be available and reserved for the kernel until the virtual to real address translation is established. After that, for the remainder of the boot and for subsequent operation of AIX, real memory up to 0.5 MB must be available and dedicated to the AIX kernel. These additional restrictions due to AIX place some limits upon a vendor's freedom in assigning memory map addresses.

Table 30. Low Memory Area Definition	
Location	**Description**
X'00000000'-X'2FFF'	PowerPC architected interrupt vector location. See *The PowerPC Architecture*, Chapter 13.
X'00003000'-X'30FF'	IBM copyright notice
X'00003100'-X'3FFF'	Usermode milicode routines and user mode kernel routines
X'00004000'-X'47FF'	Kernel patch area
X'00004800'-X'4FFF'	Kernel overlay area and kernel branch table

E.8 Hardware Abstraction Layer

/ An abstraction layer is, fundamentally, a way of structuring the operating system environment to isolate its hardware-dependent components and facilitate the introduction of changes in the processor architecture and system structure. It is an essential component of how the kernel, subsystems, device drivers, and, ultimately, applications can be made to run compatibly across the product line and from release to release. It can be used very effectively to assist in the porting of an operating system to variations of existing platforms.

/ The AIX kernel has a modular structure and a formalized and documented set of interfaces. These attributes
/ will persist across operating system releases and can achieve many of the same goals as the abstraction soft-
/ ware. Vendors who desire to differentiate their platforms for AIX must obtain a source license and then
/ modify kernel components which are affected by the hardware differences.

The introduction of any layer between the hardware and software inevitably raises the question of performance impacts. In the AIX kernel definition, we allow the performance optimizations to occur in a well-defined manner, and it is our intent to provide extensions to the interface to meet future requirements. The AIX kernel, as it is defined, enhances the portability of AIX, as we continue to implement our palmtop-to-teraflop platform vision. There are a number of other related benefits of the structured AIX kernel, such as resource utilization, product time-to-market, and maintenance and support, which are not discussed here.

From a structural view, the routines in the kernel which interface to hardware components will be provided with a set of services to perform a requested function independent of the underlying hardware. The main components of the AIX kernel fall into categories of service which account for differences in processor, I/O, and platform-specific implementation. Typical services which will be provided under this framework are memory management services (cache and DMA), I/O services (access to bus controllers and system I/O), bring-up and configuration services (hardware initialization), interrupts, device drivers, and RAS (access to NVRAM and system error registers). As an example of one of these services, in the hardware initialization routine for an operating system supporting platforms with different I/O models, static or dynamic checking of each implementation would have to be included in the routine. In the AIX environment, the routine would make a call to a kernel component that is platform specific.

/ Figure 60 conceptually shows the AIX kernel framework. The kernel hardware services make the underlying hardware accessible to the kernel, device drivers, and subsystems in an abstract manner. The service which is invoked by the routines within the system components remains the same for any number of implementation variations, and the specific processor architecture (Power or PowerPC), platform (PowerPC Reference Platform, PowerPC Reference Platform-clone or SMP), or I/O (PCI or ISA) variation would be accounted for
/ only once, in this service layer. As is shown by the links to these hardware services, there are services defined for the kernel, such as initialization and configuration; others which may be subsystem or device driver specific, such as DMA; and those that may apply to any, such as cache management.

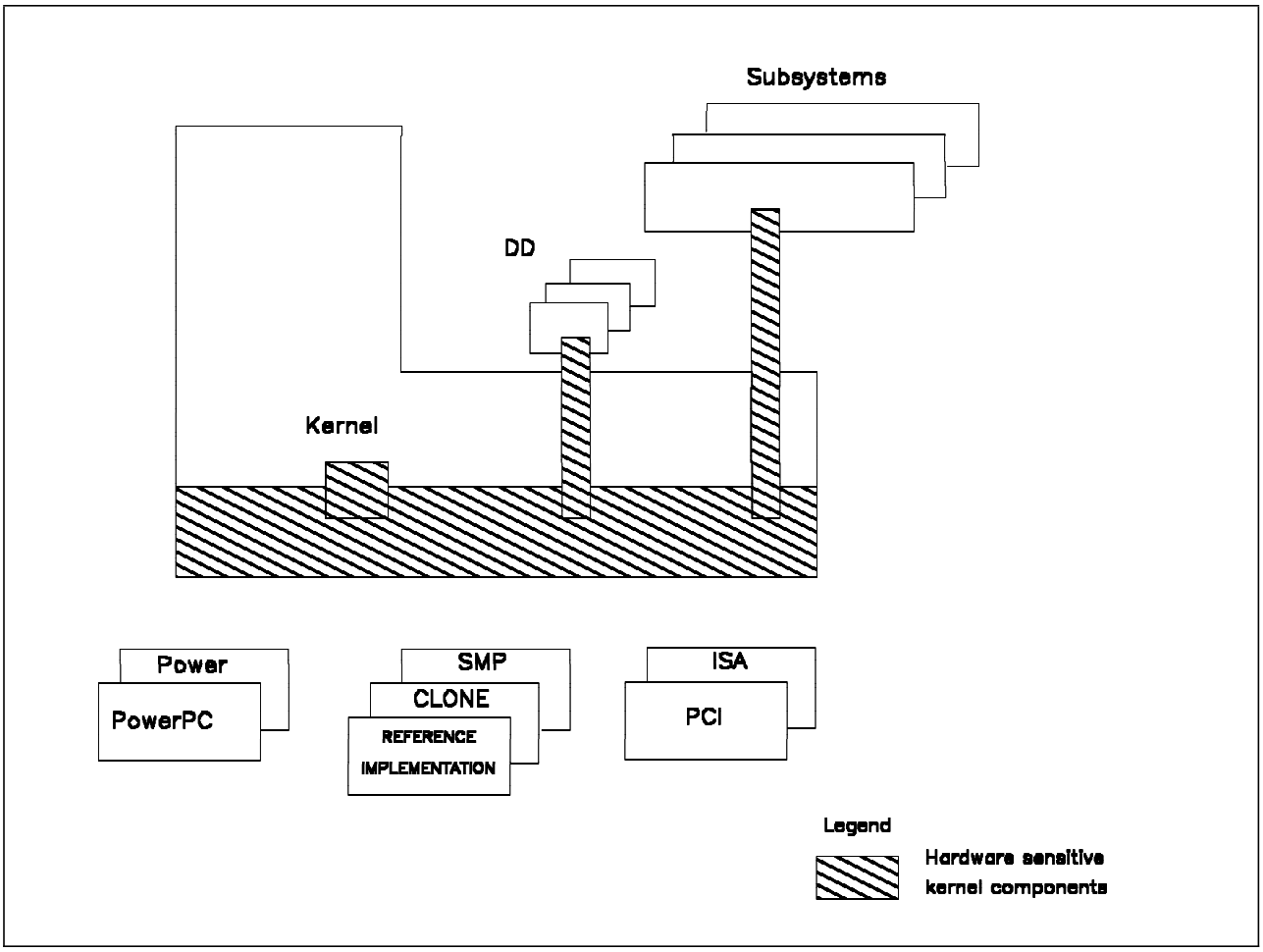


Figure 60. Structure of the AIX Kernel

E.9 Device Driver Model

A device driver is a section of code that provides support for a device. Device drivers run in a privileged state, as AIX kernel extensions, and have access to a number of functions that are unavailable to normal application programs. They shield the user from device-specific details and provide a common I/O model for accessing the devices for which they provide support.

Device drivers can play two roles in AIX: the “device head” role and the “device handler” role.

A device head is a device driver or a portion thereof that provides interfaces to application programs via the standard *open, close, read, write,* and related system calls. A device driver acting in this role takes I/O requests from application programs and communicates them to a device handler. The interface between application programs and a device head is rigidly defined by the AIX kernel itself.

A device handler is the portion of a device driver that communicates with the actual device or adaptor. The device handler takes requests from a device head and implements the request on real hardware. The interface between a device head and a device handler is essentially undefined by AIX, though a large number of primitive functions are provided by AIX to assist in constructing the interface. The details, however, are left to the device driver author. The interface between the device handler and the device itself is naturally dependent on the hardware being manipulated, though AIX again provides a set of functions which assist in performing the hardware interface.