MultiProcessor Specification

Version 1.4

August 1996

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

A license is hereby granted to copy and reproduce this specification for internal use only.

No other license, express or implied, by estoppel or otherwise, to any other intellectual property rights is granted herein.

Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information in this specification. Intel does not warrant or represent that such implementation(s) will not infringe such rights.

Intel Corporation and Intel’s FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

* Third-party trademarks are the property of their respective owners.

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Center
P.O. Box 7641
Mt. Prospect IL 60056-7641

or call 800-879-4683

printed on
recycled paper

Revision History

Revision	Revision History	Date
	Pre-release Version 1.0. Formerly called “PC+MP Specification”	10/27/93
-001	Version 1.1. Resolves conflicts with MCA-based systems. The following changes have been made: 1. Two MP feature information bytes were moved from the BIOS System Configuration Table to the RESERVED area of the MP Floating Pointer Structure. 2. If the MP Floating Pointer Structure is present, it indicates that the system is MP-compliant, in accordance with this specification. (Previously, this was indicated by bit 0 of MP Feature Information Byte 1.) 3. One more hardware default system configuration was added for MCA+PCI with the integrated APIC.	4/11/94
-002	Minor technical corrections.	6/1/94
-003	Added Appendix D: Multiple I/O APIC Multiple PCI Bus Systems.	9/1/94
-004	Version 1.4. Added extended configuration table to improve support for multiple PCI bus configurations and improve future expandability. Made clarifications to Appendix B.	7/1/95
-005	Added Appendix E: Errata. Includes information for hierarchical PCI bus.	08/15/96

Table of Contents

Chapter 1 Introduction

Chapter 2 System Overview

Chapter 3 Hardware Specification

Chapter 4 MP Configuration Table

Chapter 5 Default Configurations

Figures

Tables

Examples

1 Introduction

The MultiProcessor Specification, hereafter known as the "MP specification," defines an enhancement to the standard to which PC manufacturers design DOS-compatible systems. MP-capable operating systems will be able to run without special customization on multiprocessor systems that comply with this specification. End users who purchase a compliant multiprocessor system will be able to run their choice of operating systems.

The MP specification covers PC/AT-compatible MP platform designs based on Intel processor architectures and Advanced Programmable Interrupt Controller (APIC) architectures. The term "PC/AT-compatible" here refers to the software-visible components of the PC/AT, not to hardware features, such as the bus implementation, that are not visible to software. An implementation of this specification may incorporate one or more industry standard buses, such as ISA, EISA, MCA, PCI, or other OEM-specific buses.

1.1 Goals

The intent of this specification is to establish an MP Platform interface standard that extends the performance of the existing PC/AT platform beyond the traditional single processor limit, while maintaining 100% PC/AT binary compatibility.

The ultimate goal is to enable scalable, high-end workstations and enterprise server systems that provide computer users with superior price/performance and have the ability to execute all existing AT binaries, as well as MP-ready software packages on shrink-wrapped MP operating systems. Figure 1-1 shows that at the heart of the specification are the data structures that define the configuration of the MP system. The BIOS constructs the MP configuration data structures, presenting the hardware in a known format to the standard device drivers or to the hardware abstraction layer of the operating system. The specification details default hardware configurations, and, for added flexibility, outlines extensions to the standard BIOS.

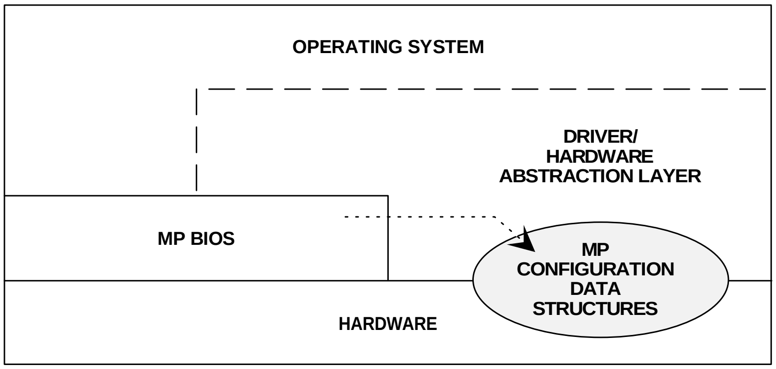


Figure 1-1. Conceptual Overview

1.2 Features of the Specification

The MP specification includes the following features:

- A multiprocessor extension to the PC/AT platform that runs all existing uniprocessor shrink-wrapped binaries, as well as MP binaries.
- Support for symmetric multiprocessing with one or more processors that are Intel architecture instruction set compatible, such as the CPUs in the Intel486™ and the Pentium® processor family.
- Support for symmetric I/O interrupt handling with the APIC, a multiprocessor interrupt controller.
- Flexibility to use a BIOS with minimal MP-specific support.
- An optional MP configuration table to communicate configuration information to an MP operating system.
- Incorporation of ISA and other industry standard buses, such as EISA, MCA, VL and PCI buses in MP-compliant systems.
- Requirements that make secondary cache and memory bus implementation transparent to software.

1.3 Scope

There are many vendors building innovative MP systems based on Intel architectures today. Intel supports and encourages vendors to develop advanced approaches to the technological requirements of today's computing environments. In no way does the MP specification prevent system manufacturers from adding their own unique value to MP systems. This specification does not define the only way that multiprocessor systems can be implemented. Vendors may, for example, create noncompliant, high-performance, scalable multiprocessor systems that do not have the PC/AT compatibility required by this specification. Hardware vendors will always have the option of offering one or more customized operating systems that support the unique, value-added capabilities that they have designed into their systems. End users will be able to benefit from the additional capabilities that these vendors offer.

The MP specification benefits PC vendors who wish to offer MP-enabled systems without having to invest in the customization of one or more operating systems. This specification focuses on providing a standard mechanism to add MP support to personal computers built around the PC/AT hardware standard. With that goal, the specification covers only the minimum set of capabilities required to extend the PC/AT platform to be MP-capable. The hardware required to implement the MP specification is kept to a minimum, as follows:

- One or more processors that are Intel architecture instruction set compatible, such as the CPUs in the Intel486 or Pentium processor family.
- One or more APICs, such as the Intel 82489DX Advanced Programmable Interrupt Controller or the integrated APIC, such as that on the Intel Pentium 735\90 and 815\100 processors, together with a discrete I/O APIC unit.
- The necessary supporting electronics to duplicate the initialization and signaling protocol described in this specification.
- PC/AT-compliant hardware.

In addition to the hardware requirements, this document also specifies MP features that are visible to the BIOS and operating system. However, it is important to understand that as hardware technology progresses, the functions performed by the BIOS may change in accordance with the hardware technology. **ONLY THE INTERFACE TO THE OPERATING SYSTEM LEVEL IS EXPECTED TO REMAIN CONSTANT.**

This specification does not address issues relating to the processor's System Management Mode (SMM).

1.4 Target Audience

This document is intended for the following users:

- OEMs who will be creating PC/AT-compatible, MP-ready hardware based on this specification.
- BIOS developers, either those who create general-purpose BIOS products or those who modify these products to suit specific MP hardware.
- Operating-system developers who will be adapting MP operating systems to run on the class of MP-ready platform specified here.

1.5 Organization of This Document

Table 1-1 shows the organization of this document.

Table 1-1. Document Organization

Chapter	Description
2	Overview of the MultiProcessor Specification
3	Specification of the MP hardware
4	Specification of MP configuration information available to OS
5	Specification of default hardware configurations
Appendix A	Guidelines for MP BIOS programming
Appendix B	Guidelines for MP operating system programming
Appendix C	Checklist for hardware compliance to the specification
Appendix D	Clarifications for multiple I/O APIC, multiple PCI bus systems
Glossary	Definitions of specialized terms used in this document

1.6 Conventions Used in This Document

Signal names that are followed by the character # represent active low signals. For example, FERR# is active when at its low-voltage state.

Throughout this document, the Intel 82489DX APIC is referred to as the “discrete APIC.” The term “integrated APIC” is used to refer to an APIC integrated with other system components, such as the Pentium 735\90 and 815\100 processors. This specification uses the term APIC to refer to both discrete and integrated versions.

The processors of the Intel486 and Pentium processor family are “little endian” machines. This means that the low-order byte of a multibyte data item in memory is at the lowest address, while the high-order byte is at the highest address. Illustrations of data structures in memory show the lowest addresses at the bottom and the highest addresses at the top of the illustration, as shown in Figure 1-2. Bit positions are numbered from right to left.

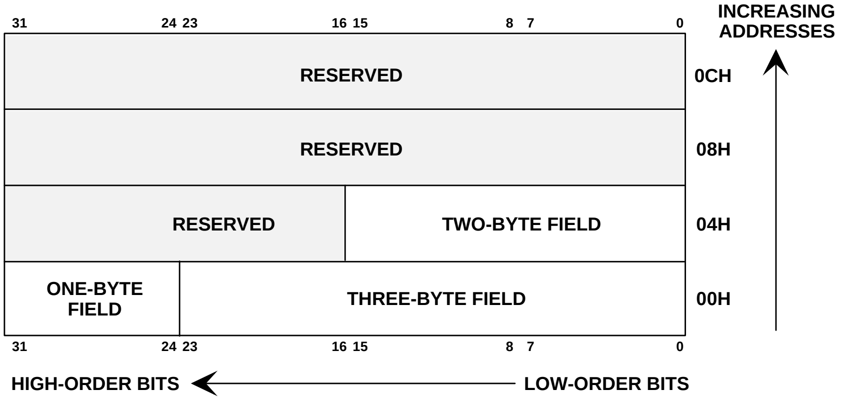


Figure 1-2. Memory Layout Conventions

In some memory layout descriptions, certain fields are marked **RESERVED**. Software should initialize these fields as binary zeros, but should otherwise treat them as having a future, though unknown effect. **SOFTWARE SHOULD AVOID ANY DEPENDENCE ON THE VALUES IN THE RESERVED FIELDS.**

1.7 For More Information

For more information, refer to any of the following documents:

- *82489DX Advanced Programmable Interrupt Controller* (data book), Intel order number 290446
- *Intel486 Microprocessor Programmer's Reference Manual,* Intel order number 240486
- *Intel Processor Identification with the CPUID Instruction* AP-485, Intel order number 241618
- *Pentium Processor User’s Manual*, Intel order number 241428

2
System Overview

In the realm of multiprocessor architectures, there are several conceptual models for tying together computing elements, and there are a variety of interconnection schemes and details of implementation. Figure 2-1 shows the general structure of a design based on the MP specification. The MP specification's model of multiprocessor systems incorporates a tightly-coupled, shared-memory architecture with a distributed interprocessor and I/O interrupt capability. It is fully symmetric; that is, all processors are functionally identical and of equal status, and each processor can communicate with every other processor. There is no hierarchy, no master-slave relationship, no geometry that limits communication only to "neighboring" processors. The model is symmetric in two important respects:

- *Memory symmetry*. Memory is symmetric when all processors share the same memory space and access that space by the same addresses. Memory symmetry offers a very important feature—the ability for all processors to execute a single copy of the operating system. Any existing system and application software will execute the same, regardless of the number of processors installed in a system.
- *I/O symmetry*. I/O is symmetric when all processors share access to the same I/O subsystem (including I/O ports and interrupt controllers) and any processor can receive interrupts from any source. Some multiprocessor systems that have symmetric access to memory are actually asymmetric with regard to I/O interrupts, because they dedicate one processor to interrupt functions. I/O symmetry helps eliminate the potential of an I/O bottleneck, thereby increasing system scalability.

With both memory and I/O symmetry, a system that complies with the MP specification can achieve hardware scalability as well as support software standardization. Based on this kind of scalable architecture, systems developers can produce systems that span a wide range of price and performance, and that all execute the same binaries.

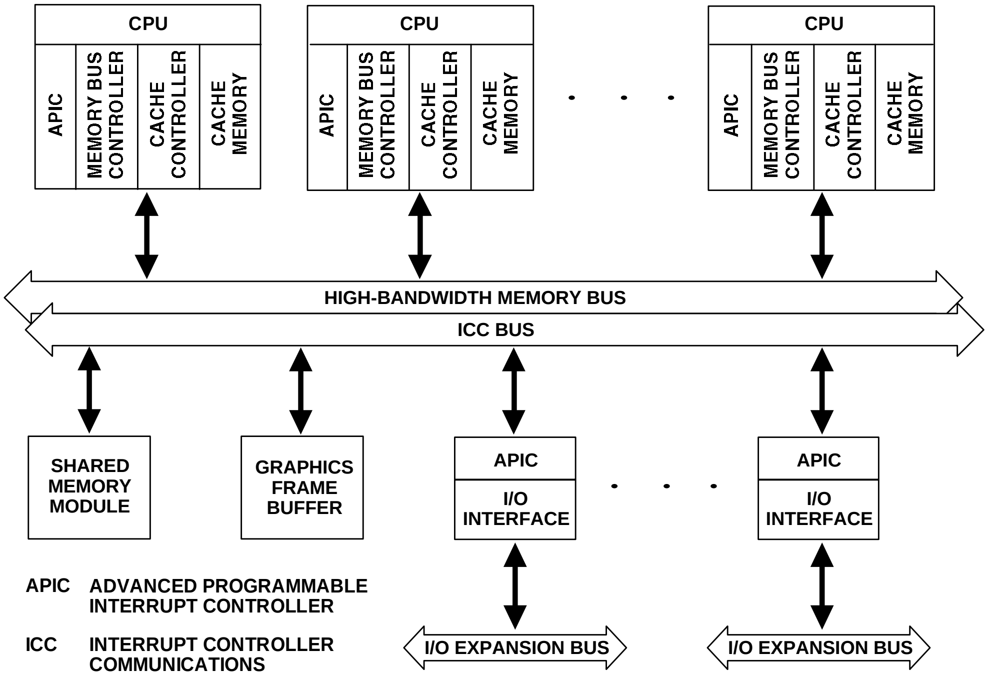


Figure 2-1. Multiprocessor System Architecture

2.1 Hardware Overview

The MP specification defines a system architecture based on the following hardware components:

- One or more processors that are Intel architecture instruction set compatible, such as the CPUs in the Intel486 and the Pentium processor family.
- One or more APICs, such as the Intel 82489DX Advanced Programmable Interrupt Controller or the integrated APIC on the Pentium 735\90 and 815\100 processors.
- Software-transparent cache and shared memory subsystem.
- Software-visible components of the PC/AT platform.

2.1.1 System Processors

To maintain compatibility with existing PC/AT software products, this specification is based on the Intel486 and the Pentium processor family. To achieve a minimum level of MP system performance, two or more processors that are Intel architecture instruction set compatible are required.

Figure 2-2 gives a different point of view of a compliant system, showing the configuration of the APICs with respect to the CPUs. While all processors in a compliant system are functionally identical, this specification classifies them into two types: the bootstrap processor (BSP) and the application processors (AP). Which processor is the BSP is determined by the hardware or by the hardware in conjunction with the BIOS. This differentiation is for convenience and is in effect only during the initialization and shutdown processes. The BSP is responsible for initializing the system and for booting the operating system; APs are activated only after the operating system is up and running. CPU1 is designated as the BSP. CPU2, CPU3, and so on, are designated as the APs.

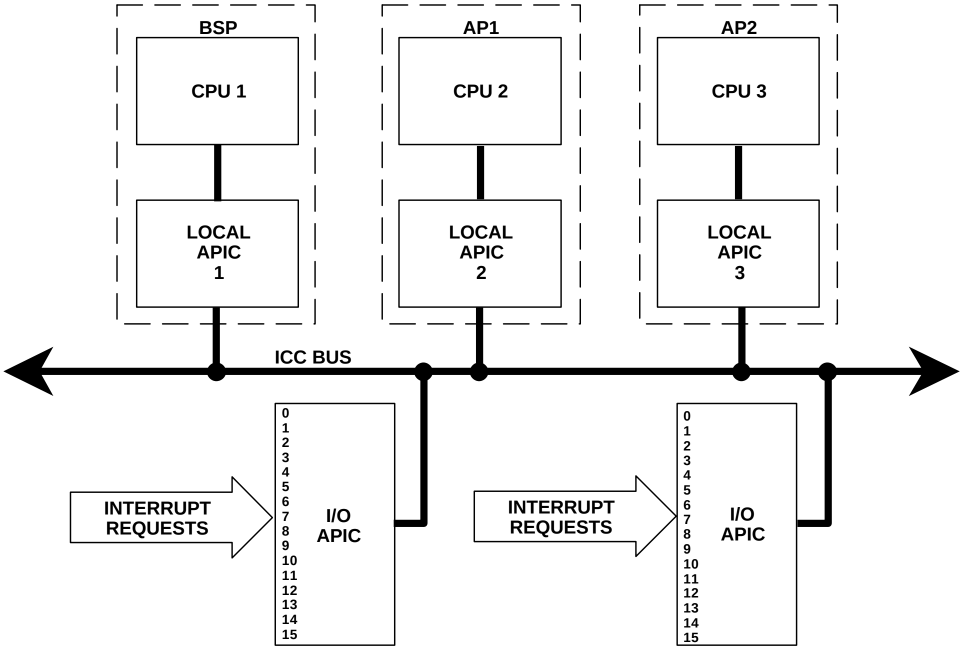


Figure 2-2. APIC Configuration

2.1.2 Advanced Programmable Interrupt Controller

The Advanced Programmable Interrupt Controller (APIC) is based on a distributed architecture in which interrupt control functions are distributed between two basic functional units, the local unit and the I/O unit. The local and I/O units communicate through a bus called the Interrupt Controller Communications (ICC) bus, as shown in Figure 2-2.

In a multiprocessor system, multiple local and I/O APIC units operate together as a single entity, communicating with one another over the ICC bus. The APIC units are collectively responsible for delivering interrupts from interrupt sources to interrupt destinations throughout the multiprocessor system.

The APICs help achieve the goal of scalability by doing the following:

- Off-loading interrupt-related traffic from the memory bus, making the memory bus more available for processor use.
- Helping processors share the interrupt processing load with other processors.

The APICs help achieve the goal of AT-compatibility by cooperating with 8259A-equivalent PICs in the system.

The local APIC units also provide interprocessor interrupts (IPIs), which allow any processor to interrupt any other processor or set of processors. There are several types of IPIs. Among them, the INIT IPI and the STARTUP IPI are specifically designed for system startup and shutdown.

Each local APIC has a Local Unit ID Register and each I/O APIC has an I/O Unit ID Register. The ID serves as a physical name for each APIC unit. It is used by software to specify destination information for I/O interrupts and interprocessor interrupts, and is also used internally for accessing the ICC bus.

Due to the distributed architecture, the APIC local and I/O units can be implemented in either a single chip, such as Intel’s 82489DX interrupt controller, or they can be integrated with other parts of the system’s components. For example, the local APIC may be integrated with the CPU chip, such as Intel’s Pentium processors (735\90, 815\100), and the I/O APIC may be integrated with the I/O chipset, such as Intel’s 82430 PCI-EISA bridge chipset.

2.1.3 System Memory

A system that complies with the MP specification uses the standard AT memory architecture. All memory is allocated for system memory with the exception of addresses 0A_0000h through 0F_FFFFh and 0FFFE_0000h through 0FFFF_FFFFh, which are reserved for I/O devices and the BIOS.

Compared to a uniprocessor system, a symmetric multiprocessor system imposes a high demand for memory bus bandwidth. The demand is proportional to the number of processors on the memory bus. To reduce memory bus bandwidth limitations, an implementation of this specification should use a secondary cache that has high-performance features, such as a write-back update policy and a snooping cache-consistency protocol. A secondary cache can push the scalability limit upward by reducing bus traffic and increasing bus bandwidth.

While both secondary caches and memory bus controllers are critical components for high performance in a symmetric multiprocessor system, this specification does not detail their implementation, requiring only that they be totally software transparent.

2.1.4 I/O Expansion Bus

The MP specification provides a multiprocessor extension to the industry-standard PC/AT platform. The term “industry-standard PC/AT platform” here refers to the software-visible components of the PC/AT. The standard does not designate a specific bus architecture. All industry-standard buses, such as ISA, EISA, MCA, VL, and PCI, can be incorporated in the design. Systems developers can implement one or more bus types in their designs, depending on the systems’ I/O performance or capacity requirements.

2.2 BIOS Overview

A BIOS functions as an insulator between the hardware on one hand, and the operating system and applications software on the other. A standard uniprocessor BIOS performs the following functions:

- Tests system components.
- Builds configuration tables to be used by the operating system.
- Initializes the processor and the rest of the system to a known state.
- Provides run-time device-oriented services.

For a multiprocessor system, the BIOS may perform the following additional functions:

- Pass configuration information to the operating system that identifies all processors and other multiprocessing components of the system.
- Initialize all processors and the rest of the multiprocessing components to a known state.

This specification allows a wide range of capability in the BIOS. At the minimal end of the capability scale, the system developer can simply insert an MP floating pointer structure in the standard BIOS. The cost of this level of simplicity in the BIOS, however, is that the system developer has less flexibility in the design of the hardware. At the maximal end of the BIOS capability scale might be a BIOS that dynamically configures the system to provide resilience in the face of component malfunctions.

BIOS developers should read Chapters 3, 4, 5, and Appendix A to understand the tradeoffs between hardware and BIOS capabilities.

2.3 Operating System Overview

Enabling the creation of shrink-wrapped MP operating systems is one of the principal goals of this specification. This goal is achieved by allowing a flexible balance between the capabilities of the hardware and those of the BIOS. The potentially vast variety of hardware configurations is reduced by the BIOS to a few simple scenarios that can be readily handled by the low-level boot-up phase of the operating system.

Operating-system developers should read Chapters 3, 4, and 5, and Appendixes A and B to fully understand the interface between the BIOS and the operating system.

3
Hardware Specification

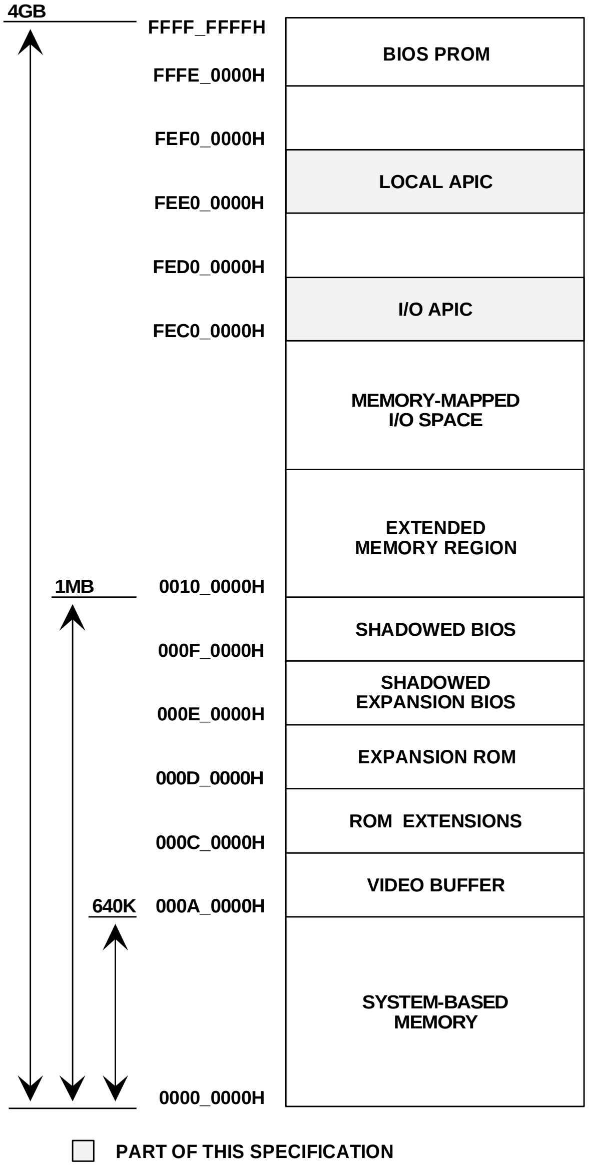
This section outlines the minimal set of common hardware features necessary for the operating system to operate on multiple hardware platforms. The MP hardware specification defines how the components mentioned in Chapter 2 are implemented. Compliance to the specification involves the following aspects of hardware implementation:

- System memory configuration
- System memory cacheability and shareability
- External cache implementation requirements
- Control of memory contention (locking)
- Ordering of posted memory writes
- Multiprocessor interrupt control
- Reset support
- Interval timer usage
- Support for fault-resilient booting

While the bulk of the MP hardware specification pertains to multiprocessor interrupt control, other areas also require some attention. The following sections take up each of these topics in turn.

3.1 System Memory Configuration

The MP memory specifications are based on the standard PC/AT memory map, which currently has a physical memory space of four gigabytes, as shown in see Figure 3-1. Physical memory should begin at 0 and be contiguous. Memory-mapped I/O devices should be mapped at the top of physical memory. The APIC default base memory addresses defined by this specification are 0FEC0_0000h and 0FEE0_0000h.



UNSHADED ADDRESS REGIONS ARE FOR REFERENCE ONLY AND SHOULD NOT BE CONSTRUED AS THE SOLE DEFINITION OF A PC/AT-COMPATIBLE ADDRESS SPACE.

Figure 3-1. System Memory Address Map

3.2 System Memory Cacheability and Shareability

The cacheability and shareability of the physical memory space are defined in Table 3-1. The address space reserved for the local APIC is used by each processor to access its own local APIC. The address space reserved for the I/O APIC must be shareable by all processors to permit dynamic reconfiguration.

Table 3-1. Memory Cacheability Map

Addresses (in hex)	Size	Description	Shared by All Processors?	Cacheable?	Comment
0000_0000h – 0009_FFFFh	640KB	Main memory	Yes	Yes	
000A_0000h – 000B_FFFFh	128KB	Display buffer for video adapters	Yes	No	
000C_0000h – 000D_FFFFh	128KB	ROM BIOS for add-on cards	Yes	Yes	
000E_0000h – 000F_FFFFh	128KB	System ROM BIOS	Yes	Yes	
0010_0000h – 0FEBF_FFFFh		Main memory	Yes	Yes	Maximum address depends on total memory installed in the system.
Not specified.		Memory-mapped I/O devices	Yes[2]	Not specified	Top unused memory
0FEC0_0000h – 0FECF_FFFFh[1]		APIC I/O unit	Yes	No	Refer to the register description in the APIC data book.
0FED0_0000h – 0FEDF_FFFFh		Reserved for memory-mapped I/O devices	Yes[2]	Not specified	
0FEE0_0000h-0FEEF_FFFFh[1]		APIC Local Unit	No	No	Refer to the register description in the APIC data book.
0FEF0_0000h – 0FFFD_FFFFh		Reserved for memory-mapped I/O devices	Yes[2]	Not specified	
0FFFE_0000h – 0FFFF_FFFFh	128KB	Initialization ROM	Yes	Not specified	

NOTES:

1. These addresses are part of this specification. The other address regions in this table are shown for reference only, and should not be construed as the sole definition of a PC/AT-compatible address space format or cache.
2. Any memory-mapped device should be shareable unless the nature of the device is that there is one device per processor.

3.3 External Cache Subsystem

Intel-compatible processors support multiprocessing both on the processor bus and on a memory bus, both with and without secondary cache units. Due to the high bandwidth demands of multiprocessor systems, external caches are often employed to improve performance. The existence and implementation details of external caches are not a part of this specification. However, when external caches are used, they must conform to certain requirements with regard to the following design issues:

- *Maintaining cache coherency*—When one processor accesses data cached in another processor's cache, it must not receive incorrect data. If it modifies data, all other processors that access that data also must not receive stale data. External caches must maintain coherency among themselves, and with the main memory, internal caches, and other bus master DMA devices.
- *Cache flushing*—The processor can generate special flush and write-back bus cycles that must be used by external caches in a manner that maintains cache coherency. The actual responses are implementation-specific and may vary from design to design. A program can initiate hardware cache flushing by executing a WBINVD instruction. This instruction is only guaranteed to flush the caches of the local processor. See Appendix B for system-wide flushing mechanisms. Given that cache coherency is maintained by hardware, there is no need for software to issue cache flush instructions under normal circumstances.
- *Reliable communication*—All processors must be able to communicate with each other in a way that eliminates interference when more than one processor accesses the same area in memory simultaneously. The processor uses the LOCK# signal for this purpose. External caches must ensure that all locked operations are visible to other processors.
- *Write ordering*—In some circumstances, it is important that memory writes be observed externally in precisely the same order as programmed. External write buffers must maintain the write ordering of the processor.

3.4 Locking

To protect the integrity of certain critical memory operations, Intel-compatible processors provide an output signal called LOCK#. For any given memory access, LOCK# is asserted once, but may remain asserted for as many memory bus cycles as required to complete the memory operation. It is the responsibility of the system hardware designers to use this signal to control memory accesses among processors.

A compliant system in multiprocessor mode must guarantee atomicity of locked-aligned memory operations; however, the implementation is not specified in this specification. A compliant system must lock at least the area of memory defined by the destination operand. A specific implementation may lock a broader area—it may even lock the entire bus. Therefore, software must consider this behavior.

To guarantee AT compatibility, locking of misaligned memory operations over other AT-compatible buses in the compliant system must be strictly implemented in accordance with the bus specifications. A compliant system may not be required to support the misaligned memory operations over its internal shared memory bus, if it is AT compatible. Operating system and software developers must ensure that data is aligned if locked access is required, because lock operations on misaligned data are not guaranteed to work on all platforms.

3.5 Posted Memory Write

When controlling I/O devices, it is important that memory and I/O operations be carried out in the order programmed. Intel-compatible processors do not buffer I/O writes; thus, strict ordering among I/O operations is enforced by the processors.

To optimize memory performance, processors and chipsets often implement write buffers and writeback caches. Intel-compatible processors guarantee processor ordering on all internal cache and write buffer accesses. However, chipsets must also guarantee processor ordering on all external memory accesses.

For systems based on the integrated APIC, posting of memory writes may result in spurious interrupts for memory-mapped I/O devices using level-triggered interrupts. I/O device drivers must serialize instructions to ensure that the device interrupt clear command reaches the device before the EOI command reaches the APIC and handles the spurious interrupt in case one occurs.

3.6 Multiprocessor Interrupt Control

In an MP-compliant system, interrupts are controlled through the APIC. The following sections describe the APIC architecture and the three interrupt modes allowed in an MP-compliant system.

3.6.1 APIC Architecture

The Intel Advanced Programmable Interrupt Controller (APIC) is based on a distributed architecture. Interrupt control functions are distributed between two basic functional units: the local unit and the I/O unit. The local and I/O units communicate through a bus called the ICC bus. The I/O unit senses an interrupt input, addresses it to a local unit, and sends it over the ICC bus. The local unit that is addressed accepts the message sent by the I/O unit.

In an MP-compliant system, one local APIC per CPU is required. Depending on the total number of interrupt lines in an MP system, one or more I/O APICs may be used. The bus interrupt line assignments can be implementation-specific and can be defined by the MP configuration table described in Chapter 4.

The Intel 82489DX APIC is a “discrete APIC” implementation. The programming interface of the 82489DX APIC units serves as the base of the MP specification. Each APIC has a version register that contains the version number of a specific APIC implementation. The version register of the 82489DX family has a version number of “0*x*,” where *x* is a four-bit hexadecimal number. Version number “1*x*” refers to Pentium processors with integrated APICs, such as the Pentium 735\90 and 815\100 processors, and *x* is a four-bit hexadecimal number.

The integrated APIC maintains the same programming interface as the 82489DX APIC. Table 3-2 describes the features specific to the integrated APIC.

Table 3-2. APIC Versions

APIC Type	Local APIC Version Register (hexadecimal)	Integrated APIC Features
82489DX APIC	0x	
Integrated APIC, i.e., Pentium processors (735\90, 815\100)	1x	STARTUP IPI. See Appendix B.4.2 for details. Programmable interrupt input polarity

NOTE:

x is a 4-bit hexadecimal number.

To encourage future extendibility and innovation, the Intel APIC architecture definition is limited to the programming interface of the APIC units. The ICC bus protocol and electrical specifications are considered implementation-specific. That is, while different versions of APIC implementations may execute the same binary software, different versions of APIC components may be implemented with different bus protocols or electrical specifications. Care must be taken when using different versions of the APIC in a system.

The APIC architecture is designed to be scaleable. The 82489DX APIC has an 8-bit ID register that can address from one to 255 APIC devices. Furthermore, the Logical Destination register for the 82489DX APIC supports 32 bits, which can address up to 32 devices. For small system implementations, the APIC ID register can be reduced to the least significant 4 bits and the Logical Destination register can be reduced to the most significant 8 bits.

To ensure software compatibility with all versions of APIC implementations, software developers must follow the following programming guidelines:

1. Assign an 8-bit APIC ID starting from zero.
2. Assign logical destinations starting from the most significant byte of the 32-bit register.
3. Program the APIC spurious vector to hexadecimal “*x*F,” where *x* is a 4-bit hexadecimal number.

The following features are only available in the integrated APIC:

1. The I/O APIC interrupt input signal polarity can be programmable.
2. A new interprocessor interrupt, STARTUP IPI is defined.

In general, the operating system must use the STARTUP IPI to wake up application processors in systems with integrated APICs, but must use INIT IPI in systems with the 82489DX APIC. Refer to Appendix B, Section B.4, for application processor startup.

3.6.2 Interrupt Modes

The MP specification defines three different interrupt modes as follows:

1. *PIC Mode*—effectively bypasses all APIC components and forces the system to operate in single-processor mode.
2. *Virtual Wire Mode*—uses an APIC as a virtual wire, but otherwise operates the same as PIC Mode.
3. *Symmetric I/O Mode*—enables the system to operate with more than one processor.

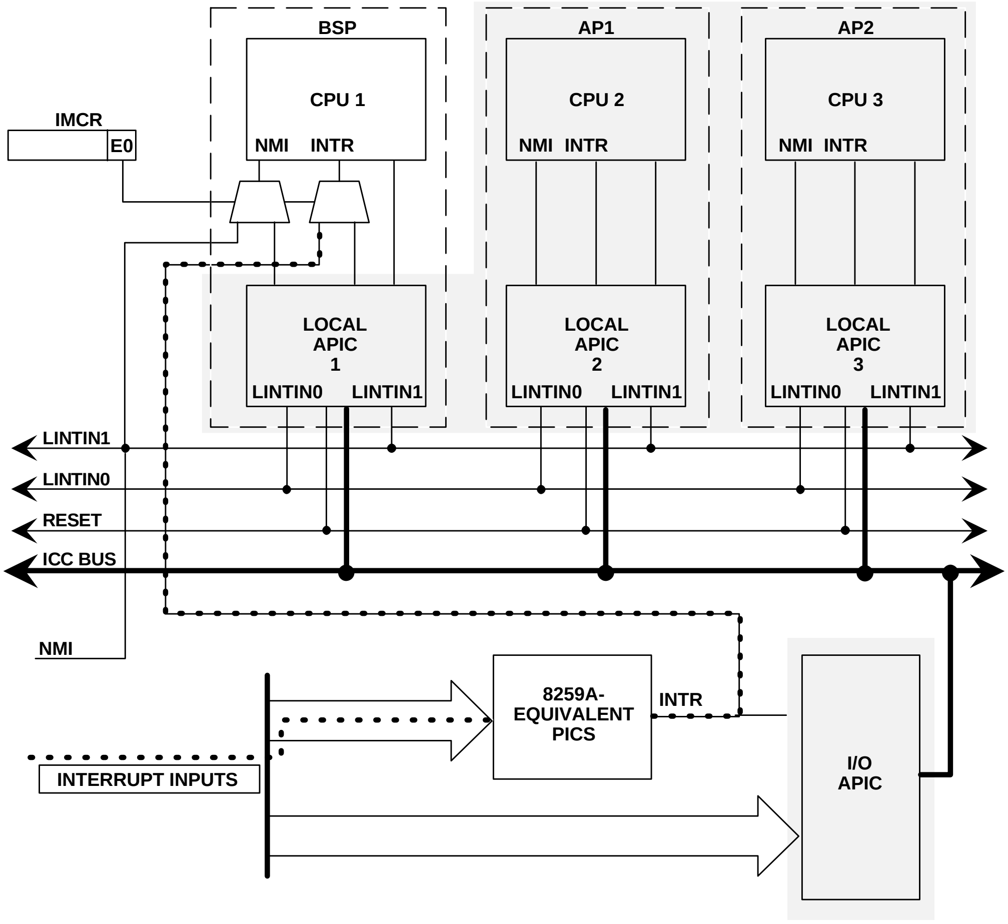
The first two interrupt modes, PIC Mode and Virtual Wire Mode, provide PC/AT-compatibility. At least one of these modes must be implemented in systems that comply with the MP specification. In these modes, full DOS compatibility with the uniprocessor PC/AT is provided by using the APICs in conjunction with standard 8259A-equivalent programmable interrupt controllers (PICs).

The third mode, Symmetric I/O Mode, must be implemented in addition to either PIC Mode or Virtual Wire Mode. An MP operating system is booted under either one of the two PC/AT-compatible modes. Later the operating system switches to Symmetric I/O Mode as it enters multiprocessor mode.

The interrupt modes are implemented by a combination of hardware and software. The hardware and programming specifications for each of these modes are further defined in the following subsections. BIOS programmers should refer to Appendix A, which includes information about programming the APIC for Virtual Wire Mode. Operating system programmers should refer to Appendix B, which provides more information about initializing the APIC for Symmetric I/O Mode.

3.6.2.1 PIC Mode

PIC Mode is software compatible with the PC/AT because it actually employs the same hardware interrupt configuration. As Figure 3-2 illustrates, the hardware for PIC Mode bypasses the APIC components by using an interrupt mode configuration register (IMCR). This register controls whether the interrupt signals that reach the BSP come from the master PIC or from the local APIC. Before entering Symmetric I/O Mode, either the BIOS or the operating system must switch out of PIC Mode by changing the IMCR.



SHADED AREAS INDICATE UNUSED CIRCUITS. DOTTED LINE SHOWS INTERRUPT PATH.

Figure 3-2. PIC Mode

The IMCR is supported by two read/writable or write-only I/O ports, 22h and 23h, which receive address and data respectively. To access the IMCR, write a value of 70h to I/O port 22h, which selects the IMCR. Then write the data to I/O port 23h. The power-on default value is zero, which connects the NMI and 8259 INTR lines directly to the BSP. Writing a value of 01h forces the NMI and 8259 INTR signals to pass through the APIC.

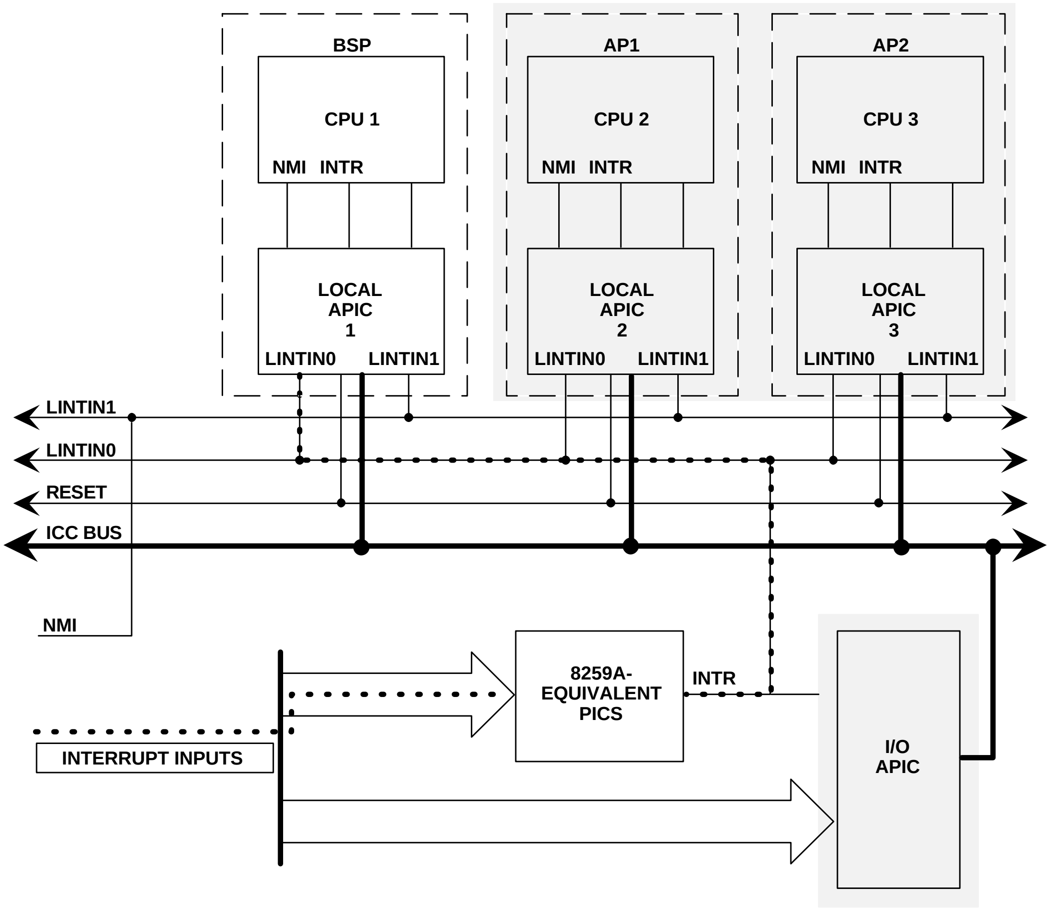
The IMCR must be cleared after a system-wide INIT or RESET to enable the PIC Mode as default. (Refer to Section 3.7 for information on the INIT and RESET signals.)

The IMCR is optional if PIC Mode is not implemented. The IMCRP bit of the MP feature information bytes (refer to Chapter 4) enables the operating system to detect whether the IMCR is implemented.

3.6.2.2 Virtual Wire Mode

Virtual Wire Mode provides a uniprocessor hardware environment capable of booting and running all DOS shrink-wrapped software.

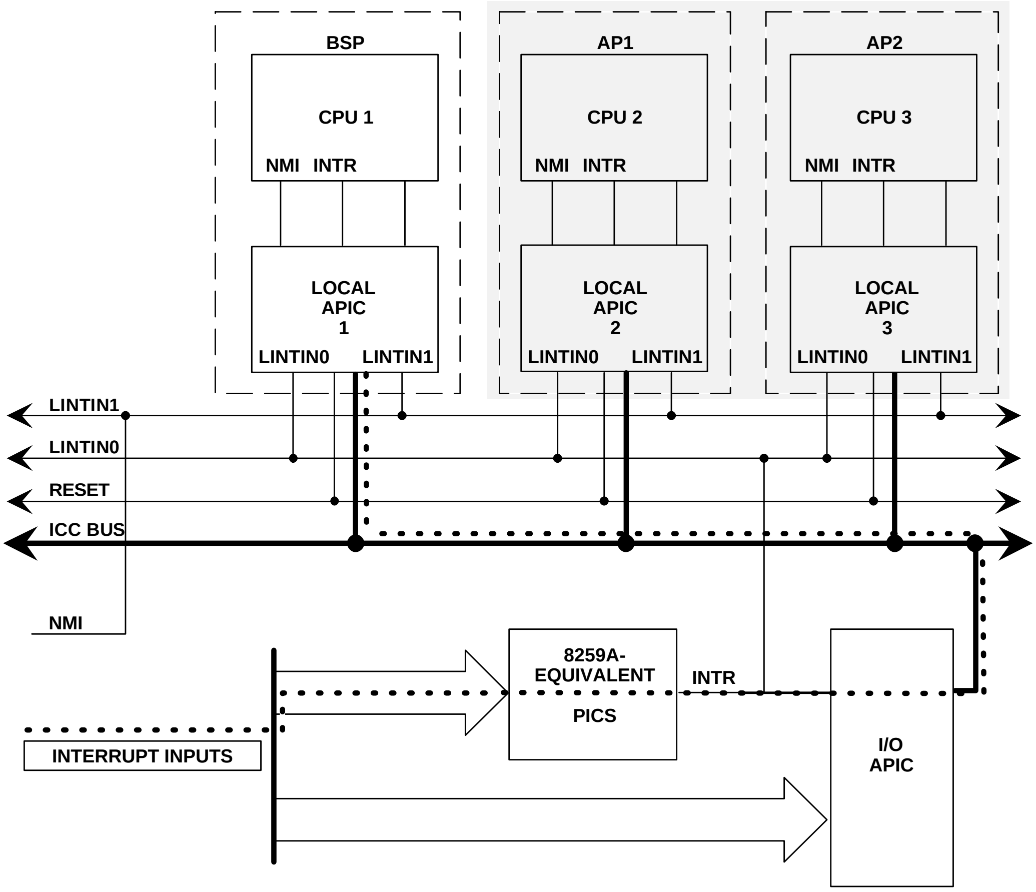
In Virtual Wire Mode, as shown in Figure 3-3, the 8259A-equivalent PIC fields all interrupts, and the local APIC of the BSP becomes a virtual wire, which delivers interrupts from the PIC to the BSP via the local APIC's local interrupt 0 (LINTIN0). The LINTIN0 pin of the local APIC is programmed as ExtINT, specifying to the APIC that the PIC is to serve as an external interrupt controller. Whenever the local APIC finds that a particular interrupt is of type ExtINT, it asserts the ExtINTA transaction along with the PINT interrupt to the processor. In this case, the I/O APIC is not used.



SHADED AREAS INDICATE UNUSED CIRCUITS. DOTTED LINE SHOWS INTERRUPT PATH.

Figure 3-3. Virtual Wire Mode via Local APIC

Figure 3-3 shows how Virtual Wire Mode can be implemented through the BSP's local APIC. It is also permissible to program the I/O APIC for Virtual Wire Mode, as shown in Figure 3-4. In this case the interrupt signal passes through both the I/O APIC and the BSP's local APIC.

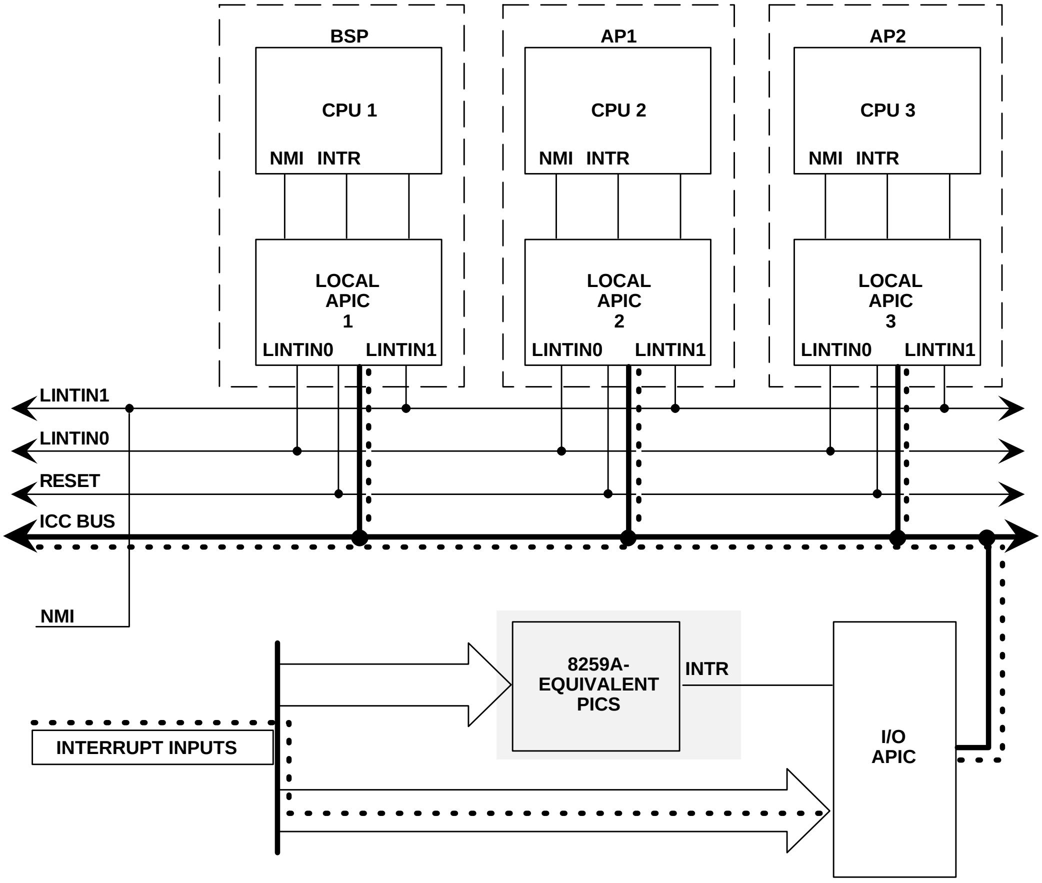


SHADED AREAS INDICATE UNUSED CIRCUITS. DOTTED LINE SHOWS INTERRUPT PATH.

Figure 3-4. Virtual Wire Mode via I/O APIC

3.6.2.3 Symmetric I/O Mode

Some MP operating systems operate in Symmetric I/O Mode. This mode requires at least one I/O APIC to operate. In this mode, I/O interrupts are generated by the I/O APIC. All 8259 interrupt lines are either masked or work together with the I/O APIC in a mixed mode. See Figure 3-5 for an overview of Symmetric I/O Mode.



SHADED AREAS INDICATE UNUSED CIRCUITS. DOTTED LINE SHOWS INTERRUPT PATH.

Figure 3-5. Symmetric I/O Mode

The APIC I/O unit has general-purpose interrupt inputs that can be individually programmed to different operating modes. The I/O APIC interrupt line assignments are system implementation specific. Refer to Chapter 4 for custom implementations and to Chapter 5 for default configurations.

The hardware must support a mode of operation in which the system can switch easily to Symmetric I/O mode from PIC or Virtual Wire mode. When the operating system is ready to switch to MP operation, it writes a 01H to the IMCR register, if that register is implemented, and enables I/O APIC Redirection Table entries. The hardware must not require any other action on the part of software to make the transition to Symmetric I/O mode.

3.6.3 Assignment of System Interrupts to the APIC Local Unit

The APIC local unit has two general-purpose interrupt inputs, which are reserved for system interrupts. These interrupt inputs can be individually programmed to different operating modes. Like the I/O APIC interrupt lines, the local APIC interrupt line assignments of a non-PC/AT-compatible system are system implementation specific. Refer to Chapter 4 for custom implementations and to Chapter 5 for default configurations.

3.6.4 Floating Point Exception Interrupt

For PC/AT compatibility, the bootstrap processor must support DOS-compatible FPU execution and exception handling while running in either of the PC/AT-compatible modes. This means that floating point error signals from the BSP must be routed to the interrupt request 13 signal, IRQ13, when the system is in PIC or virtual wire mode. While floating point error signals from an application processor need not be routed to IRQ13, platform designers may choose to connect the two. For example, connecting the floating point error signal from application processors to IRQ13 can be useful in the case of a platform that supports dynamic choice of BSP during boot.

In symmetric mode, a compliant system supports only on-chip floating point units, with error signaling via interrupt vector 16. Operating systems must use interrupt vector 16 to manage floating point exceptions when the system is in symmetric mode.

It is recommended that hardware platforms be designed to block delivery of floating point exception signals from the processors once the system has switched into symmetric mode to avoid delivery of superfluous interrupts. If done, such blocking must be implemented in a manner that is transparent to the operating system. However, the operating system must still be prepared to handle interrupts generated through assertion of floating point error signals, because on some platforms these signals may still be routed to IRQ13 even after the switch to symmetric mode.

3.6.5 APIC Memory Mapping

In a compliant system, all APICs must be implemented as memory-mapped I/O devices. APIC base addresses are at the top of the memory address space. All APIC local units are mapped to the same addresses, which are not shared. Each processor accesses its local APIC via these memory addresses. The default base address for the local APICs is 0FEE0_0000h.

Unlike the local APICs, the I/O APICs are mapped to give shared access from all processors, providing full symmetric I/O access. The default base address for the first I/O APIC is 0FEC0_0000h. Subsequent I/O APIC addresses are assigned in 4K increments. For example, the second I/O APIC is at 0FEC0_1000h.

Non-default APIC base addresses can be used if the MP configuration table is provided. (Refer to Chapter 4.) However, the local APIC base address must be aligned on a 4K boundary, and the I/O APIC base address must be aligned on a 1K boundary.

3.6.6 APIC Identification

Systems developers must assign APIC local unit IDs and ensure that all are unique. There are two acceptable ways to assign local APIC IDs, as follows:

- By hardware. The ID of each APIC local unit is sampled from the appropriate pins at RESET.
- By the BIOS. Software can override the APIC IDs assigned by hardware by writing to the Local Unit ID Register. This is possible only with help from the hardware; for example, using board ID registers that the software can read to determine which board has the BSP.

Local APIC IDs must be unique, and need not be consecutive.

The MP operating system can use the local APIC ID to determine on which processor it is executing.

The ID of each I/O APIC unit is set to zero during RESET. It is the responsibility of the operating system to verify the uniqueness of the I/O APIC ID and to assign a unique ID if a conflict is found. The assignment of APIC IDs for I/O units must always begin from the lowest number that is possible after the assignment of local APIC IDs. The operating system must not attempt to change the ID of an APIC I/O unit if the preset ID number is acceptable.

3.6.7 APIC Interval Timers

The 82489DX APIC local unit contains a 32-bit wide programmable timer with the following two independent clock input sources:

1. The CLK pin provides the clock signal that drives the 82489DX APIC’s internal operation.
2. The TMBASE pin allows an independent clock signal to be connected to the 82489DX APIC for use by the timer functions.

The interval timers of the integrated APIC have only one clock input source, CLK. To maintain consistency, developers of compliant systems based on the 82489DX must choose CLK as the source of the 82489DX APIC timer clock. TMBASE must be left disabled. An MP operating system may use the IRQ8 real-time clock as a reference to determine the actual APIC timer clock speed.

Special consideration must be made for systems with stop clock (STPCLK#) capability. Timer interrupts are ignored while STPCLK# is asserted. The system time-of-day clock may need to be reset when STPCLK# is deasserted.

3.6.8 Multiple I/O APIC Configurations

Systems may provide more than one I/O APIC for increased interrupt scalability in Symmetric I/O Mode. To preserve PC/AT compatibility in PIC or Virtual Wire mode, interrupts connected to additional I/O APICs must also be connected to the 8259A programmable interrupt controller. Figure 3-6 represents one possible interrupt connection scheme for a system with two I/O APICs.

The non-ISA interrupts are connected to both the 8259A IRQ inputs and the inputs of the associated I/O APIC. To prevent non-ISA interrupts from appearing at inputs on both I/O APICs, the hardware must provide a means to disable the interrupt routing network when the system switches to symmetric I/O mode with the second I/O APIC enabled.

More information about the implementation of multiple I/O APIC configurations is presented in Appendix D.

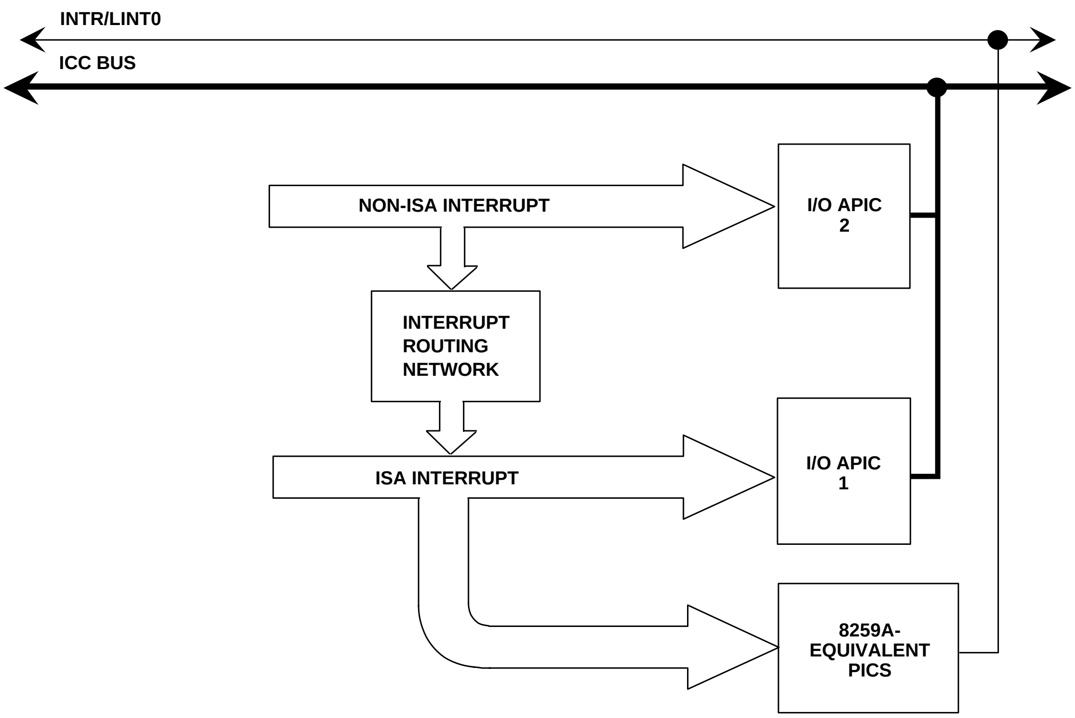


Figure 3-6. Multiple I/O APIC Configurations

3.7 RESET Support

To bring all circuitry in a computer system to an initial state, computer systems require a system-wide reset capability. To support multiple processors, a compliant system requires a processor-specific reset or initialization capability in addition to the typical system-wide reset and initialization capabilities.

- The term “RESET” refers to the system-wide hard reset. It refers to the RESET signal on both Pentium and Intel486 processors or the RESET signal of the 82489DX APIC. (See Section 3.7.1.)
- The term “INIT,” refers to either a system-wide soft reset/initialization or a processor-specific initialization. For example, the term “INIT” may refer to the INIT signal on the Pentium processor or to the RESET signal on the Intel486 processor. (See Sections 3.7.2. and 3.7.3.)

Because the INIT signal is available on the Pentium processor but not on the Intel486 processor, the remainder of this document uses the above-mentioned special definitions for the terms “INIT” and “RESET”:

3.7.1 System-wide RESET

The system-wide RESET, as defined by this specification, refers to a *hard* or *cold reset*, which sets all circuitry, including processor, coprocessor, add-in cards, and control logic, to an initial state. A hard reset is the reset signal that is sent to all components of the system during a power-on or by the front panel reset button (if the system is so equipped). This type of reset operates without regard to cycle boundaries, and, for example, is connected to the RESET pin of Pentium processors.

3.7.2 System-wide INIT

The system-wide INIT, as defined by this specification, refers to a *soft* or *warm reset* that initializes only portions of the processor. This type of reset initializes the microprocessor in such a way that the reset does not corrupt any pending cycles, but waits instead for a cycle boundary, and does not invalidate the contents of caches and floating point registers. This type of reset request is connected to the INIT signal of newer processors, such as the Pentium processors. On Intel486 processors, the RESET pin is used for this function, as well as for hard resets, but the RESET pin does not provide the advantages of the INIT pin. There are many possible ways to assert a soft reset, including:

- A write either to a port of the 8042 Keyboard Controller or to some other port provided for the same purpose by a chipset.
- A shutdown special bus cycle. Usually a chipset senses a shutdown cycle and asserts a soft reset to the processor.

In a compliant system, the standard PC/AT-platform resets mentioned above, both hard and soft, must be directed to all processors in the system, except in the case of fault-tolerant MP systems, in which a soft reset may be handled on a per-processor basis.

3.7.3 Processor-specific INIT

A processor-specific INIT is one of the basic multiprocessor support functions of a compliant multiprocessor system, along with processor startup and shutdown. With it, the BSP can selectively initialize an AP for subsequent startup or recover an AP from a fatal system error. This type of INIT function is exclusively used by the MP operating system or BIOS self-test routine. The system must be designed so that the processor-specific INIT can be initiated by software programming; it is not necessary that it be initiated by hardware.

A compliant system supports the processor-specific INIT via a special interprocessor interrupt (IPI) mechanism called INIT IPI. For the 82489DX APIC, INIT IPI is an IPI that has the delivery mode RESET, which delivers the signal to all processors listed in the destination by asserting/deasserting the addressed APIC local unit’s PRST output pin. When the PRST signal is connected to the INIT pin of the Pentium processor or to the RESET pin of the Intel486 processor, the INIT IPI forces the processor to begin executing at the reset vector.

For systems based on the Intel486 processor, the 82489DX APIC’s PRST line must be the only line connected to the processor’s RESET input, so that the INIT IPI resets the targeted processor only. The system reset signal is connected to the local 82489DX APIC’s RESET input. Assertion of the system reset signal then causes all of the local 82489DX APICs to assert their PRST outputs, thereby resetting all the processors.

For integrated APIC versions of the Pentium processor, INIT IPI asserts and deasserts the internal INIT signal of the Pentium processor.

3.8 System Initial State

The system initial state is the state before the BIOS gives control to the operating system. It is identical to the system initial state of a typical PC/AT system, with the additional MP components in the following state:

1. All local APICs are disabled, except for the local APIC of the BSP if the system starts in Virtual Wire Mode.
2. All pending I/O APIC interrupts are cleared and disabled.
3. If IMCR is present, it should be set to 0 or 1 depending on the interrupt mode chosen for startup.
4. All APs are in Real Mode.
5. All APs are in HALT state or off the system bus.

The BIOS must disable interrupts to all processors and set the APICs to the system initial state before giving control to the operating system. The operating system is responsible for initializing all of the APICs.

3.9 Support for Fault-resilient Booting

OEMs may choose not to implement a fault-resilient booting capability in their systems. However, if such a capability is provided, systems developers must observe the following guidelines:

- BSP determination may be performed by special hardware or by the BIOS, but it must be totally transparent to the operating system.
- NMI and INTR must be connected to the BSP.
- FERR# and IGNNE# signals from the designated BSP must be used to support IRQ13.
- The A20M# signal from the designated BSP must be used to support the masking of physical address bit 20 on the BSP (to support DOS compatibility).

4
MP Configuration Table

The operating system must have access to some information about the multiprocessor configuration. The MP specification provides two methods for passing this information to the operating system: a minimal method for configurations that conform to one of a set of common hardware defaults, and a maximal method that provides the utmost flexibility in hardware design. Figure 4-1 shows the general layout of the MP configuration data structures.

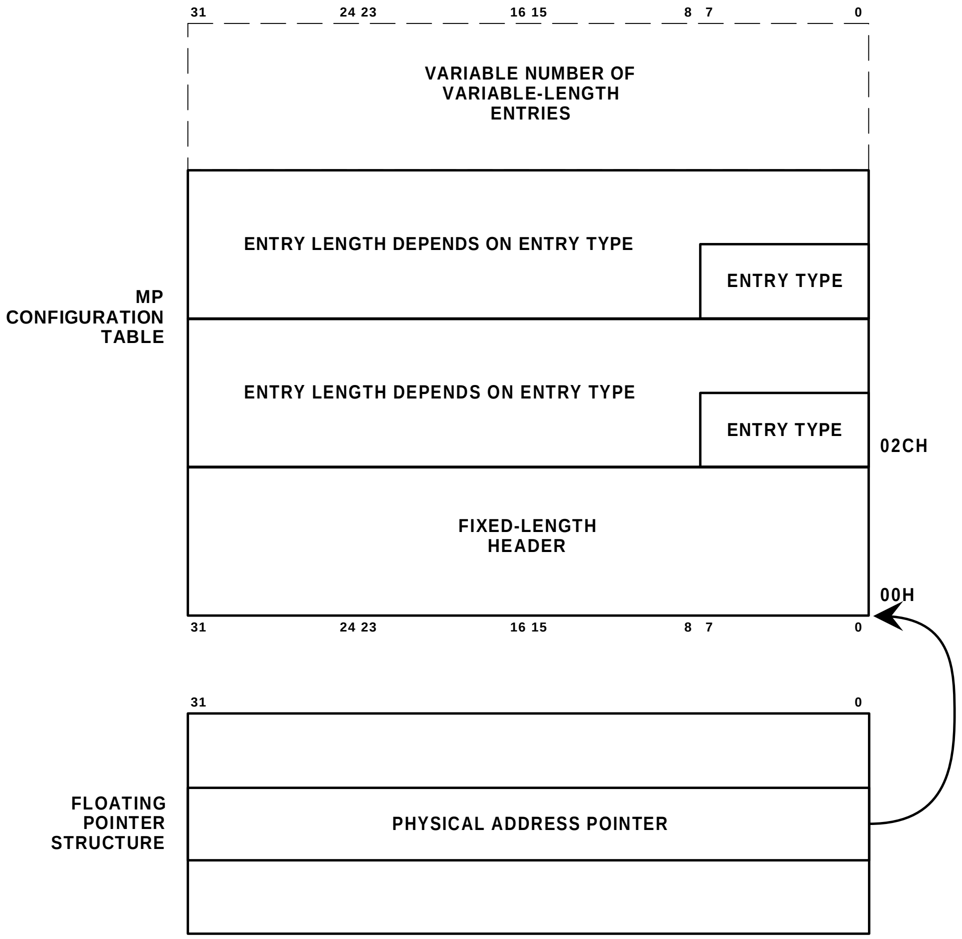


Figure 4-1. MP Configuration Data Structures

The following two data structures are used:

1. The *MP Floating Pointer Structure*. This structure contains a physical address pointer to the MP configuration table and other MP feature information bytes. When present, this structure indicates that the system conforms to the MP specification. This structure must be stored in at least one of the following memory locations, because the operating system searches for the MP floating pointer structure in the order described below:
   a. In the first kilobyte of Extended BIOS Data Area (EBDA), or
   b. Within the last kilobyte of system base memory (e.g., 639K-640K for systems with 640 KB of base memory or 511K-512K for systems with 512 KB of base memory) if the EBDA segment is undefined, or
   c. In the BIOS ROM address space between 0F0000h and 0FFFFFh.
2. The *MP Configuration Table*. This table is optional. The table is composed of a base section and an extended section. The base section contains entries that are completely backwards compatible with previous versions of this specification. The extended section contains additional entry types. The MP configuration table contains explicit configuration information about APICs, processors, buses, and interrupts. The table consists of a header, followed by a number of entries of various types. The format and length of each entry depends on its type. When present, this configuration table must be stored either in a non-reported system RAM or within the BIOS read-only memory space.

An MP configuration table is not required if the system design corresponds to one of the default configurations listed in Chapter 5. Note that these defaults are only for designs that are always equipped with two processors. Systems that support a variable number of installed processors must supply a complete MP configuration table that indicates the correct number of installed processors. For example, systems that ship with only one processor but provide for a later upgrade with a second processor may not use a default MP configuration.

The following is a list of the suggested memory spaces for the MP configuration table:

a. In the first kilobyte of Extended BIOS Data Area (EBDA), or
b. Within the last kilobyte of system base memory if the EBDA segment is undefined, or
c. At the top of system physical memory, or
d. In the BIOS read-only memory space between 0E0000h and 0FFFFFh.

The BIOS reports the base memory size in a two-byte location (40:13h) of the BIOS data area. The base memory size is reported in kilobytes minus 1K, which is used by the EBDA segment or for other purposes.

The exact starting address of the EBDA segment for EISA or MCA systems can be found in a two-byte location (40:0Eh) of the BIOS data area. If system memory is used, the BIOS must not report this memory as part of the available memory space.

These two MP configuration data structures can be located in the ROM space only if the system is not dynamically reconfigurable. The MP configuration information is intended to be read-only to the operating system.

Strings in the configuration tables are coded in ASCII. The first character of the string is stored at the lowest address of the string field. If the string is shorter than its field, the extra character locations are filled with space characters. Strings are not null terminated.

4.1 MP Floating Pointer Structure

An MP-compliant system must implement the MP floating pointer structure, which is a variable length data structure in multiples of 16 bytes. Currently, only one 16-byte data structure is defined. It must span a minimum of 16 contiguous bytes, beginning on a 16-byte boundary, and it must be located within the physical address as specified in the previous section. To determine whether the system conforms to the MP specification, the operating system must search for the MP floating pointer structure in the order specified in the previous section. Figure 4-2 shows the format of this structure, and Table 4-1 explains each of the fields.

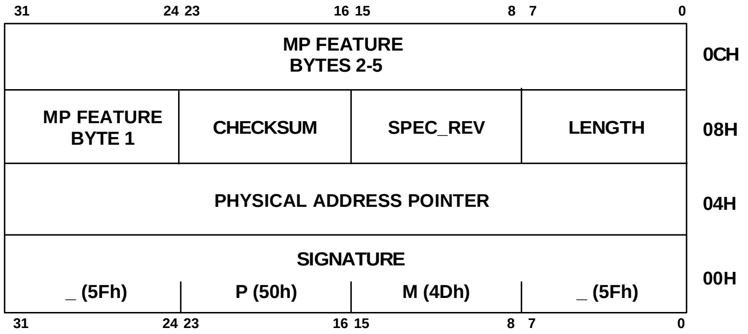


Figure 4-2. MP Floating Pointer Structure

Table 4-1. MP Floating Pointer Structure Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
SIGNATURE	0	32	The ASCII string represented by "_MP_" which serves as a search key for locating the pointer structure.
PHYSICAL ADDRESS POINTER	4	32	The address of the beginning of the MP configuration table. All zeros if the MP configuration table does not exist.
LENGTH	8	8	The length of the floating pointer structure table in paragraph (16-byte) units. The structure is 16 bytes or 1 paragraph long; so this field contains 01h.
SPEC_REV	9	8	The version number of the MP specification supported. A value of 01h indicates Version 1.1. A value of 04h indicates Version 1.4.
CHECKSUM	10	8	A checksum of the complete pointer structure. All bytes specified by the length field, including CHECKSUM and reserved bytes, must add up to zero.

Table 4-1. MP Floating Pointer Structure Fields (continued)

Field	Offset (in bytes:bits)	Length (in bits)	Description
MP FEATURE INFORMATION BYTE 1	11	8	**Bits 0-7:** MP System Configuration Type. When these bits are all zeros, the MP configuration table is present. When nonzero, the value indicates which default configuration (as defined in Chapter 5) is implemented by the system.
MP FEATURE INFORMATION BYTE 2	12:0 12:7	7 1	**Bits 0-6:** Reserved for future MP definitions. **Bit 7:** IMCRP. When the IMCR presence bit is set, the IMCR is present and PIC Mode is implemented; otherwise, Virtual Wire Mode is implemented.
MP FEATURE INFORMATION BYTES 3-5	13	24	Reserved for future MP definitions. Must be zero.

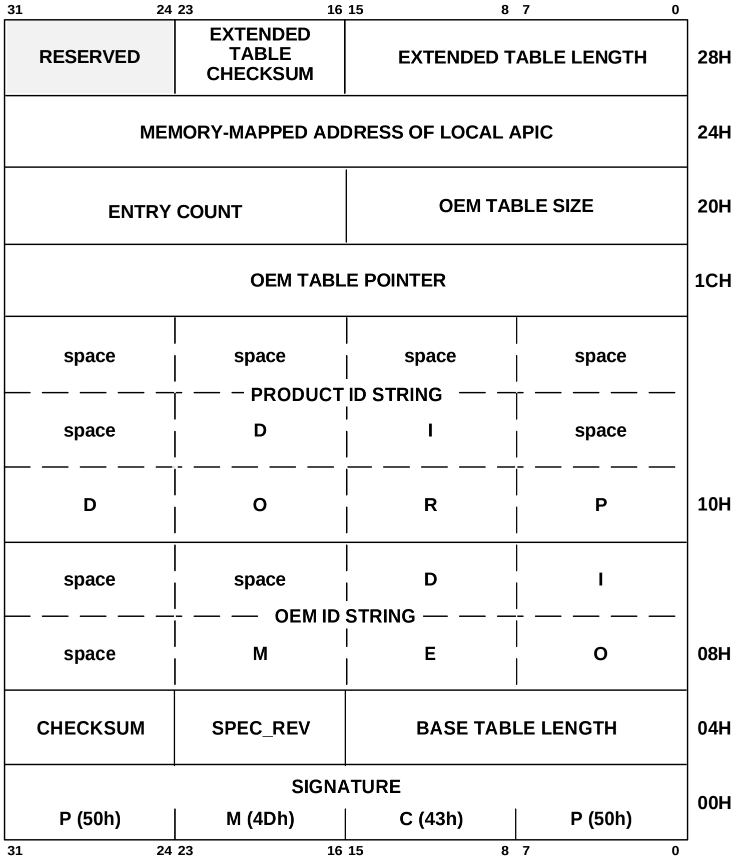
The MP feature information byte 1 specifies the MP system default configuration type. If nonzero, the system configuration conforms to one of the default configurations. The default configurations, specified in Chapter 5, may only be used to describe systems that always have two processors installed.

Bit 7 of MP feature information byte 2, the IMCR present bit, is used by the operating system to determine whether PIC Mode or Virtual Wire Mode is implemented by the system.

The physical address pointer field contains the address of the beginning of the MP configuration table. If it is nonzero, the MP configuration table can be accessed at the physical address provided in the pointer structure. This field must be all zeros if the MP configuration table does not exist.

4.2 MP Configuration Table Header

Figure 4-3 shows the format of the header of the MP configuration table, and Table 4-2 explains each of the fields.



31 24	23 16	15 8	7 0	
RESERVED	EXTENDED TABLE CHECKSUM	EXTENDED TABLE LENGTH		28H
MEMORY-MAPPED ADDRESS OF LOCAL APIC				24H
ENTRY COUNT		OEM TABLE SIZE		20H
OEM TABLE POINTER				1CH
space	space	space	space	
PRODUCT ID STRING				
space	D	I	space	
D	O	R	P	10H
space	space	D	I	
OEM ID STRING				
space	M	E	O	08H
CHECKSUM	SPEC_REV	BASE TABLE LENGTH		04H
SIGNATURE				
P (50h)	M (4Dh)	C (43h)	P (50h)	00H
31 24	23 16	15 8	7 0	

Figure 4-3. MP Configuration Table Header

Table 4-2. MP Configuration Table Header Fields

Field	Offset (in bytes)	Length (in bits)	Description
SIGNATURE	0	32	The ASCII string representation of "PCMP," which confirms the presence of the table.
BASE TABLE LENGTH	4	16	The length of the base configuration table in bytes, including the header, starting from offset 0. This field aids in calculation of the checksum.
SPEC_REV	6	8	The version number of the MP specification. A value of 01h indicates Version 1.1. A value of 04h indicates Version 1.4.
CHECKSUM	7	8	A checksum of the entire base configuration table. All bytes, including CHECKSUM and reserved bytes, must add up to zero.
OEM ID	8	64	A string that identifies the manufacturer of the system hardware.
PRODUCT ID	16	96	A string that identifies the product family.
OEM TABLE POINTER	28	32	A physical-address pointer to an OEM-defined configuration table. This table is optional. If not present, this field is zero.
OEM TABLE SIZE	32	16	The size of the base OEM table in bytes. If the table is not present, this field is zero.
ENTRY COUNT	34	16	The number of entries in the variable portion of the base table. This field helps the software identify the end of the table when stepping through the entries.
ADDRESS OF LOCAL APIC	36	32	The base address by which each processor accesses its local APIC.
EXTENDED TABLE LENGTH	40	16	Length in bytes of the extended entries that are located in memory at the end of the base configuration table. A zero value in this field indicates that no extended entries are present.
EXTENDED TABLE CHECKSUM	42	8	Checksum for the set of extended table entries, including only extended entries starting from the end of the base configuration table. When no extended table is present, this field is zero.

4.3 Base MP Configuration Table Entries

A variable number of variable length entries follow the header of the MP configuration table. The first byte of each entry identifies the entry type. Each entry type has a known, fixed length. The total length of the MP configuration table depends upon the configuration of the system. Software must step through each entry in the base table until it reaches ENTRY COUNT. The entries are sorted on ENTRY TYPE in ascending order. Table 4-3 gives the meaning of each value of ENTRY TYPE.

Table 4-3. Base MP Configuration Table Entry Types

Entry Description	Entry Type Code*	Length (in bytes)	Comments
Processor	0	20	One entry per processor.
Bus	1	8	One entry per bus.
I/O APIC	2	8	One entry per I/O APIC.
I/O Interrupt Assignment	3	8	One entry per bus interrupt source.
Local Interrupt Assignment	4	8	One entry per system interrupt source.

* All other type codes are reserved.

4.3.1 Processor Entries

Figure 4-4 shows the format of each processor entry, and Table 4-4 defines the fields.

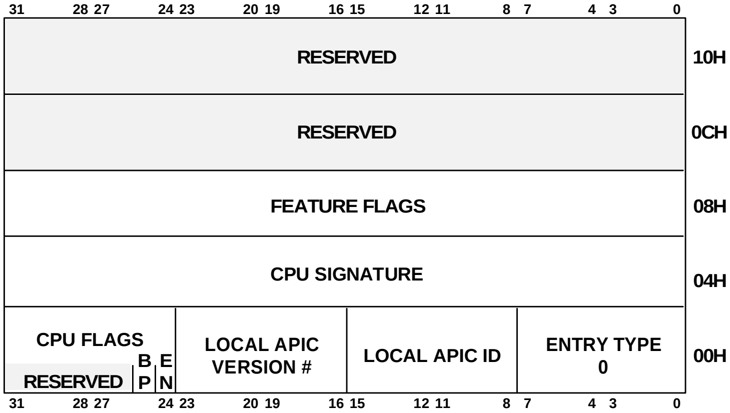


Figure 4-4. Processor Entry

In systems that use the MP configuration table, the only restriction placed on the assignment of APIC IDs is that they be unique. They do not need to be consecutive. For example, it is possible for only APIC IDs 0, 2, and 4 to be present.

Table 4-4. Processor Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	A value of 0 identifies a processor entry.
LOCAL APIC ID	1	8	The local APIC ID number for the particular processor.
LOCAL APIC VERSION #	2	8	Bits 0–7 of the local APIC's version register.
CPU FLAGS: EN	3:0	1	If zero, this processor is unusable, and the operating system should not attempt to access this processor.
CPU FLAGS: BP	3:1	1	Set if specified processor is the bootstrap processor.
CPU SIGNATURE:			
STEPPING	4:0	4	Refer to Table 4-5 for values.
MODEL	4:4	4	Refer to Table 4-5 for values.
FAMILY	5:0	4	Refer to Table 4-5 for values.
FEATURE FLAGS	8	32	The feature definition flags for the processor as returned by the CPUID instruction. Refer to Table 4-6 for values. If the processor does not have a CPUID instruction, the BIOS must assign values to FEATURE FLAGS according to the features that it detects.

The configuration table is filled in by the BIOS after it executes a CPU identification procedure on each of the processors in the system. Whenever possible, the complete 32-bit CPU signature should be filled with the values returned by the CPUID instruction. The CPU signature includes but is not limited to, the stepping, model, and family fields. If the processor does not have a CPUID instruction, the BIOS must fill these and future reserved fields with information returned by the processor in the EDX register after a processor reset. See the *Pentium Processor User's Manual* and *Intel Processor Identification with the CPUID Instruction* (AP-485) for details on the CPUID instruction.

Table 4-5. Intel486 and Pentium Processor Signatures

Family	Model	Stepping[a]	Description
0000	0000	0000	Not a valid CPU signature.
0100	0000 and 0001	xxxx	Intel486 DX Processor
0100	0010	xxxx	Intel486 SX Processor
0100	0011	xxxx	Intel487™ Processor
0100	0011	xxxx	IntelDX2™ Processor
0100	0100	xxxx	Intel486 SL Processor
0100	0101	xxxx	IntelSX2™ Processor
0100	1000	xxxx	IntelDX4™ Processor
0101	0001	xxxx	Pentium Processors (510\60, 567\66)
0101	0010	xxxx	Pentium Processors (735\90, 815\100)
	Values not shown are reserved for future processors. Refer to the documentation of each new processor for its family and model values.		
1111	1111	1111	Not a valid CPU signature. Indicates a processor that is not an Intel architecture-compatible processor (a graphics controller, for example)

[a] Intel releases information about stepping numbers as needed.

Table 4-6. Feature Flags from CPUID Instruction

Bit	Name	Description	Comments
0	FPU	On-chip Floating Point Unit	The processor contains an FPU that supports the Intel387™ processor floating point instruction set.
1–6			Reserved.
7	MCE	Machine Check Exception	Exception 18 is defined for machine checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging reporting and processor shutdowns. Machine-check exception handlers may have to depend on processor version to do model-specific processing of the exception or test for the presence of the standard machine-check feature.
8	CX8	CMPXCHG8B Instruction	The 8 byte (64 bits) compare-and-exchange instruction is supported (implicitly locked and atomic). Introduced by the Pentium processor.
9	APIC	On-chip APIC	Indicates that an integrated APIC is present and hardware enabled. (Software disabling does not affect this bit.)
10–31			Reserved.

4.3.2 Bus Entries

Bus entries identify the kinds of buses in the system. Because there may be more than one bus in a system, each bus is assigned a unique bus ID number by the BIOS. The bus ID number is used by the operating system to associate interrupt lines with specific buses. Figure 4-5 shows the format of a bus entry, and Table 4-7 explains the fields of each entry.

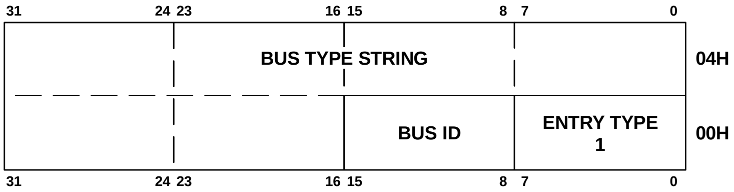


Figure 4-5. Bus Entry

Table 4-7. Bus Entry Fields

Field	Offset (in bytes)	Length (in bits)	Description
ENTRY TYPE	0	8	Entry type 1 identifies a bus entry.
BUS ID	1	8	An integer that identifies the bus entry. The BIOS assigns identifiers sequentially, starting at zero.
BUS TYPE STRING	2	48	A string that identifies the type of bus. Refer to Table 4-8 for values. These are 6-character ASCII (blank-filled) strings used by the MP specification.

Table 4-8. Bus Type String Values

Bus Type String	Description
CBUS	Corollary CBus
CBUSII	Corollary CBUS II
EISA	Extended ISA
FUTURE	IEEE FutureBus
INTERN	Internal bus
ISA	Industry Standard Architecture
MBI	Multibus I
MBII	Multibus II
MCA	Micro Channel Architecture
MPI	MPI
MPSA	MPSA
NUBUS	Apple Macintosh NuBus
PCI	Peripheral Component Interconnect
PCMCIA	PC Memory Card International Assoc.
TC	DEC TurboChannel
VL	VESA Local Bus
VME	VMEbus
XPRESS	Express System Bus

Each bus in a system must have a unique BUS ID if any one of the following criteria are true:

1. The bus does not share its memory address space with another bus.
2. The bus does not share its I/O address space with another bus.
3. The bus does not share interrupt lines with another bus.
4. Any aspect of the bus as an independent entity is software visible (such as PCI configuration space).

Special consideration must be given when assigning a BUS ID for local buses such as VL, which are designed to work in conjunction with another bus. If the bus looks like a part of another bus because it uses a subset of that bus's interrupts and address space, rendering it totally invisible to software, it does not need its own bus entry in the table. The two buses are then considered a single logical bus.

4.3.3 I/O APIC Entries

The configuration table contains one or more entries for I/O APICs. Figure 4-6 shows the format of each I/O APIC entry, and Table 4-9 explains each field.

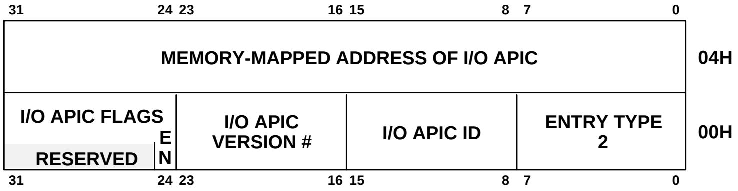


Figure 4-6. I/O APIC Entry

Table 4-9. I/O APIC Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	A value of 2 identifies an I/O APIC entry.
I/O APIC ID	1	8	The ID of this I/O APIC.
I/O APIC VERSION #	2	8	Bits 0–7 of the I/O APIC's version register.
I/O APIC FLAGS: EN	3:0	1	If zero, this I/O APIC is unusable, and the operating system should not attempt to access this I/O APIC. At least one I/O APIC must be enabled.
I/O APIC ADDRESS	4	32	Base address for this I/O APIC.

4.3.4 I/O Interrupt Assignment Entries

These entries indicate which interrupt source is connected to each I/O APIC interrupt input. There is one entry for each I/O APIC interrupt input that is connected. Figure 4-7 shows the format of each entry, and Table 4-10 explains each field. Appendix D provides the semantics for encoding PCI interrupts.

The MP specification enables significantly more interrupt sources than the standard AT architecture by using I/O APICs. When using I/O APICs, it is preferable that the buses do not share interrupts with the other buses. Bus implementations that share interrupts, such as the PCI and VL local buses, support their bus interrupts by overloading them into another bus space. These buses can be supported in one of the following two ways:

1. Interrupt Assignment Entries for each of the bus interrupts are listed in the MP configuration table. Each interrupt destination matches the destination of another interrupt source interrupt that this interrupt shares. For example, if PCI-Device1/INTA# has the same vector as ISA-IRQ2, then both Interrupt Assignment Entries for these vectors would refer to the same destination I/O APIC and INTIN#.

2. No Interrupt Assignment Entries are declared for any of the bus source interrupts, and the operating system uses some other bus-specific knowledge of bus interrupt schemes in order to support the bus. This operating system bus-specific knowledge is beyond the scope of this specification.

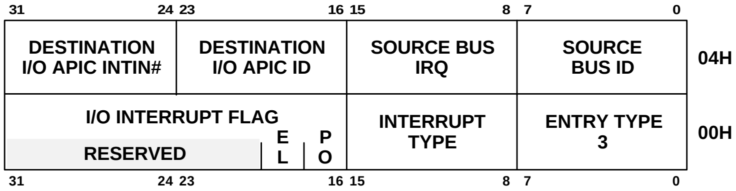


Figure 4-7. I/O Interrupt Entry

Table 4-10. I/O Interrupt Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	Entry type 3 identifies an I/O interrupt entry.
INTERRUPT TYPE	1	8	See Table 4-11 for values.
PO	2:0	2	Polarity of APIC I/O input signals: 00 = Conforms to specifications of bus (for example, EISA is active-low for level-triggered interrupts) 01 = Active high 10 = Reserved 11 = Active low Must be 00 if the 82489DX is used.
EL	2:2	2	Trigger mode of APIC I/O input signals: 00 = Conforms to specifications of bus (for example, ISA is edge-triggered) 01 = Edge-triggered 10 = Reserved 11 = Level-triggered
SOURCE BUS ID	4	8	Identifies the bus from which the interrupt signal comes.
SOURCE BUS IRQ	5	8	Identifies the interrupt signal from the source bus. Values are mapped onto source bus signals, starting from zero. A value of 0, for example, would indicate IRQ0 of an ISA bus. See Section D.3 for PCI bus semantics.
DESTINATION I/O APIC ID	6	8	Identifies the I/O APIC to which the signal is connected. If the ID is 0FFh, the signal is connected to all I/O APICs.
DESTINATION I/O APIC INTIN#	7	8	Identifies the INTIN*n* pin to which the signal is connected.

Table 4-11. Interrupt Type Values

Interrupt Type*	Description	Comments
0	INT	Signal is a vectored interrupt; vector is supplied by APIC redirection table.
1	NMI	Signal is a nonmaskable interrupt.
2	SMI	Signal is a system management interrupt.
3	ExtINT	Signal is a vectored interrupt; vector is supplied by external PIC. For example, if an 8259 is used as the external PIC, the source is the 8259 INTR output line, and the vector is supplied by the 8259.

* All other values are reserved.

4.3.5 Local Interrupt Assignment Entries

These configuration table entries tell what interrupt source is connected to each local interrupt input of each local APIC. Figure 4-8 shows the format of each entry, and Table 4-12 explains each field.

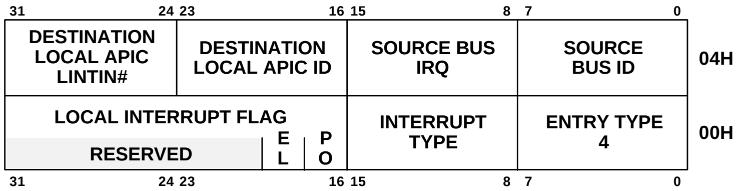


Figure 4-8. Local Interrupt Entry

Table 4-12. Local Interrupt Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	Entry type 4 identifies a local interrupt entry.
INTERRUPT TYPE	1	8	See Table 4-11 for values
PO	2:0	2	Polarity of APIC local input signals: 00 = Conforms to specifications of bus (for example, EISA is active-low for level triggered interrupts) 01 = Active high 10 = Reserved 11 = Active low Must be 00 if the 82489DX is used.
EL	2:2	2	Trigger mode of APIC local input signals: 00 = Conforms to specifications of bus (for example, ISA is edge triggered) 01 = Edge-triggered 10 = Reserved 11 = Level-triggered
SOURCE BUS ID	4	8	Identifies the bus from which the interrupt signal came.
SOURCE BUS IRQ	5	8	Identifies the interrupt signal from the source bus. Values are mapped onto source bus signals, starting from zero. A value of 0, for example, would indicate IRQ0 of an ISA bus. See Section D.3 for PCI bus semantics.
DESTINATION LOCAL APIC ID	6	8	Identifies the local APIC to which the signal is connected. If the ID is 0FFh, the signal is connected to all local APICs.
DESTINATION LOCAL APIC LINTIN#	7	8	Identifies the LINTIN*n* pin to which the signal is connected, where n = 0 or 1.

4.4 Extended MP Configuration Table Entries

A variable number of variable-length entries are located in memory, immediately following entries in the base section of the MP configuration table described in Section 4.3. These entries compose the extended section of the MP configuration table. Each entry in the extended section of the table has three elements:

- ENTRY TYPE
- ENTRY LENGTH
- DATA

ENTRY TYPE and ENTRY LENGTH make up a header that is present in all extended configuration table entries. The ENTRY TYPE and ENTRY LENGTH fields are eight bit unsigned values. ENTRY LENGTH specifies the total length of the given configuration entry. The length of the DATA field is two bytes less than the value specified by ENTRY LENGTH.

This scheme ensures that operating systems can parse all the entries in the extended MP configuration table area, even if an entry type is unrecognized. Operating systems that find an unknown entry type when parsing this section of the table should ignore the content and move on to the next entry, until the offset from the end of the base table reaches the length that is specified in the EXTENDED TABLE LENGTH field of the configuration table header. The ability to skip entries with unrecognized type codes beyond those listed in Table 4-13 is essential since it is anticipated that more types of entries will be added to this list over time.

The total length of the Extended MP configuration table depends upon the configuration of the system. The entries are sorted on ENTRY TYPE in ascending order. Table 4-13 gives the meaning of each value of ENTRY TYPE.

Table 4-13. Extended MP Configuration Table Entry Types

Entry Description	Entry Type Code*	Length (in bytes)	Comments
SYSTEM ADDRESS SPACE MAPPING	128	20	Entry to declare system visible memory or I/O space on a bus.
BUS HIERARCHY DESCRIPTOR	129	8	Entry to describe I/O bus interconnection.
COMPATIBILITY BUS ADDRESS SPACE MODIFIER	130	8	Entry to describe predefined address ranges that modify the memory or I/O space visible on a bus in order to support ISA compatibility.

*All other type codes are reserved.

4.4.1 System Address Space Mapping Entries

System Address Space Mapping entries define the system addresses that are visible on a particular bus. Each bus defined in the Base Table can have any number of System Address Space Mapping entries included in the Extended Table. Thus, individual buses can be configured to support different address ranges, thereby decreasing the amount of bus traffic on a given bus and increasing the overall system performance. Each consecutive address range that is usable by the operating system to access devices on a given bus has a System Address Space Mapping entry. Figure 4-9 shows the format of each entry, and Table 4-14 explains each field. See also Appendix E, for more information.

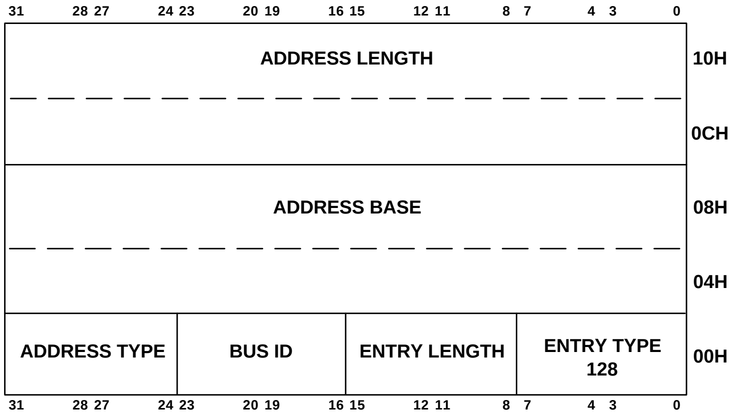


Figure 4-9. System Address Space Entry

Table 4-14. System Address Space Mapping Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	Entry type 128 identifies a System Address Space Mapping Entry.
ENTRY LENGTH	1	8	A value of 20 indicates that an entry of this type is twenty bytes long.
BUS ID	2	8	The BUS ID for the bus where the system address space is mapped. This number corresponds to the BUS ID as defined in the base table bus entry for this bus.
ADDRESS TYPE	3	8	System address type used to access bus addresses must be: 0 = I/O address 1 = Memory address 2 = Prefetch address All other numbers are reserved.
ADDRESS BASE	4	64	Starting address
LENGTH	12	64	Number of addresses which are visible to the bus

If any main memory address is mapped to a software visible bus, such as PCI, it must be explicitly declared using a System Address Space Mapping entry.

In the case of a bus that is directly connected to the main system bus, system address space records and compatibility base address modifiers must be provided as needed to fully describe the complete set of addresses that are mapped to that bus. For example, in Figure 4-10, complete explicit descriptions must be provided for PCI BUS 0 and PCI BUS 1 even if one of the buses is programmed for subtractive decode.

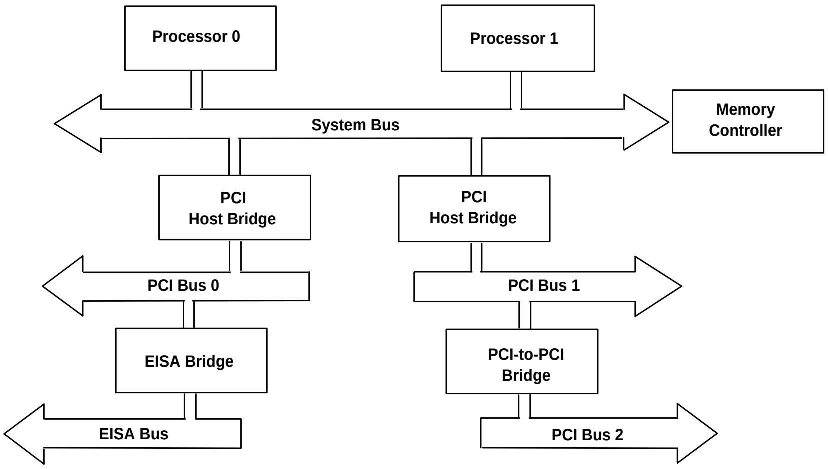


Figure 4-10. Example System with Multiple Bus Types and Bridge Types

Since all device settings must fall within supported System Address Space mapping for a given bus in order to be usable by the operating system, buses that do not support dynamically configurable devices (i.e., ISA, EISA) should support all possible addresses to that bus.

In general, the MP configuration table must provide entries to describe system address space mappings for all I/O buses present in the system. There are two exceptions to this rule:

1. For buses that are connected via PCI-to-PCI-bridge-specification-compliant bridges: in this case, the system address space mappings for such buses may be omitted from the configuration table. In such cases, the operating system is expected to discover the address space mapping by querying the PCI-to-PCI-bridge-specification-compliant bridge directly.
2. For buses that are connected via a parent I/O bus and for which the subtractive decode bit is set (refer to Section 4.4.2 for details).

Typically, this would mean that a minimal description of resources allocated to PCI buses in a system need only include System Address Space Mapping entries for PCI bus zero and any additional peer PCI buses (if present) where these buses are connected by PCI bridges that are specific to the chipset or host bus.

Note

System Address Mappings are unidirectional in nature. They only describe system addresses that propagate to the target bus from any given processor. For DMA support on the target bus, all memory addresses that contain real memory should be accessible directly by either the bus or by a bus-specific DMA controller. For buses with fewer than 32-bit address lines, all real memory at addresses that the bus can generate must be accessible for DMA.

4.4.2 Bus Hierarchy Descriptor Entry

If present, Bus Hierarchy Descriptor entries define how I/O buses are connected relative to each other in a system with more than one I/O bus. Bus Hierarchy Descriptors are used to supplement System Address Mapping entries to describe how addresses propagate to particular buses in systems where address decoding cannot be completely described by System Address Space Mapping entries alone. Entries of this type are required for each bus that is connected to the system hierarchically below another I/O bus. For example, given the system described in Figure 4-10, bus hierarchy entries are required for the EISA bus and the PCI BUS 2 since both have parent buses that are themselves I/O buses.

The Bus Hierarchy entry provides information about where in a hierarchical connection scheme a given bus is connected and the type of address decoding performed for that bus. Figure 4-11 shows the format of each entry, and Table 4-15 explains each field. See also Appendix E, for more information.

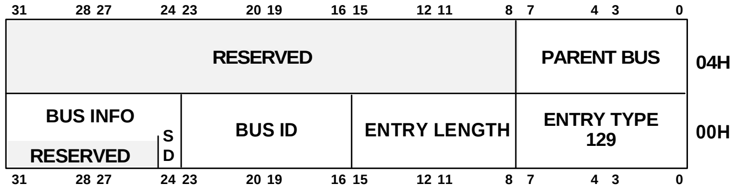


Figure 4-11. Bus Hierarchy Descriptor Entry

Table 4-15 Bus Hierarchy Descriptor Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	Entry type 129 identifies a Bus Hierarchy Descriptor Entry.
ENTRY LENGTH	1	8	A value of 8 indicates that this entry type is eight bytes long.
BUS ID	2	8	The BUS ID identity of this bus. This number corresponds to the BUS ID as defined in the base table bus entry for this bus.
BUS INFORMATION:SD	3:0	1	Subtractive Decode Bus. If set, all addresses visible on the parent bus but not claimed by another device on the parent bus (including bridges to other buses) are useable on this bus.
PARENT BUS	4	8	Parent Bus. This number corresponds to the BUS ID as defined in the base table bus entry for the parent bus of this bus

For buses where the BUS INFORMATION:SD bit is set, System Address Mappings may not be needed. Since the bus is defined as being subtractive decode, the range of addresses that appear on the bus can be derived from address decoding information for parent and peer buses.

4.4.3 Compatibility Bus Address Space Modifier Entry

The Compatibility Bus Address Space Modifier defines a set of predefined address ranges that should either be added or removed from the supported address map ranges for a given bus. This entry type is used in combination with System Address Space Mapping entries to complete the description of memory and I/O ranges that are visible on a bus that incorporates support for ISA device compatibility.

For example, a host bus bridge for a PCI bus that provides ISA compatibility may decode a predefined range of addresses used for ISA device support in addition to the addresses used for PCI devices on that bus. A Compatibility Bus Address Space Modifier can be used in this case to add these predefined address ranges to the list specified by System Address Space Mapping entries for that PCI bus. As a corollary, in a system where two peer PCI buses are included, one of which provides ISA compatibility, a Compatibility Bus Address Space Modifier can be used to subtract these predefined ranges from the address space assigned to the PCI bus that does not support ISA devices to avoid any potential conflict.

The same effect can be achieved by using System Address Space Mapping entries to completely describe the address ranges supported on a bus, including those ranges that might otherwise be described by a Compatibility Bus Address Space Modifier entry. However, given the number of discrete address ranges that are used for ISA device compatibility, using an approach based solely on System Address Space Mapping entries may result in a significantly larger number of configuration table entries and a corresponding increase in table size.

Figure 4-12 shows the format of each entry, and Table 4-16 explains each field.

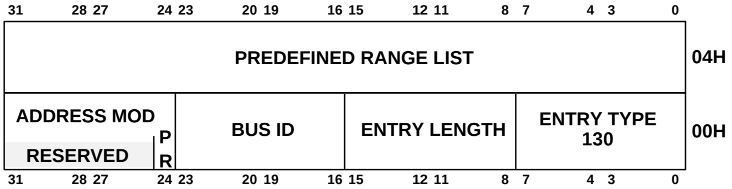


Figure 4-12. Compatibility Bus Address Space Modifier Entry

Table 4-16. Compatibility Bus Address Space Modifier Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	Entry type 130 identifies a Compatibility Bus Address Space Modifier Entry.
ENTRY LENGTH	1	8	A value of 8 indicates that an entry of this type is eight bytes long.
BUS ID	2	8	Bus for address space mappings are to be modified. This number corresponds to the BUS ID as defined in the base table entry for this bus.
ADDRESS MODIFIER:PR	3:0	1	If this bit is set to one, the address ranges specified by PREDEFINED RANGE LIST are to be subtracted from the address space associated with the bus. If this bit is set to zero, the specified address ranges are to be added to the address space associated with the bus.
PREDEFINED RANGE LIST	4	32	A number that indicates the list of predefined address space ranges that this record will modify for the bus.

PREDEFINED RANGE LIST may take one of the values from Table 4-17. The value of PREDEFINED RANGE LIST indicates the set of address ranges that are to be either added to or subtracted from the address range associated with the BUS ID.

Table 4-17. Predefined Range Lists

List	Value	Address Ranges
ISA Compatible I/O Range	0	X100-X3FF X500-X7FF X900-XBFF XD00-XFFF
VGA Compatible I/O Range	1	X3B0 - X3BB X3C0 - X3DF X7B0 - X7BB X7C0 - X7DF XBB0 - XBBB XBC0 - XBDF XFB0 - XFBB XFC0 - XFDF

All addresses in Table 4-17 are in hexadecimal notation. In each case the X represents any hexadecimal digit, 0-F. As a result, the ISA Compatible I/O Range describes 64 distinct ranges.

5
Default Configurations

The MP specification defines several default MP system configurations. The purpose of these defaults is to simplify BIOS design. If a system conforms to one of the default configurations, the BIOS will not need to provide the MP configuration table. The operating system will have the default MP configuration table predefined internally.

Default system configuration types are defined by MP feature information byte 1, which is part of the MP floating pointer structure. The physical address pointer field of the MP floating pointer structure must be zero if one of the default configurations is selected. (Refer to Chapter 4 for a detailed description of the MP floating pointer structure.)

To use a default configuration, a system must meet the following basic criteria:

1. The system supports two processors.
2. Both processors must execute the common Intel architecture instruction set.
3. The local APICs are located at base memory address 0FEE0_0000h.
4. The local APIC IDs are assigned consecutively by hardware starting from zero.
5. An I/O APIC is present at base memory address 0FEC0_0000h.
6. Either PIC Mode or Virtual Wire Mode is implemented as the power-on default interrupt mode.

The default system configurations include configurations that use the discrete APIC, such as the Intel 82489DX or its equivalent, and configurations that use the integrated APIC, such as the Pentium processors (735\90, 815\100).

Each default configuration has a unique code. Table 5-1 specifies the configuration associated with each code.

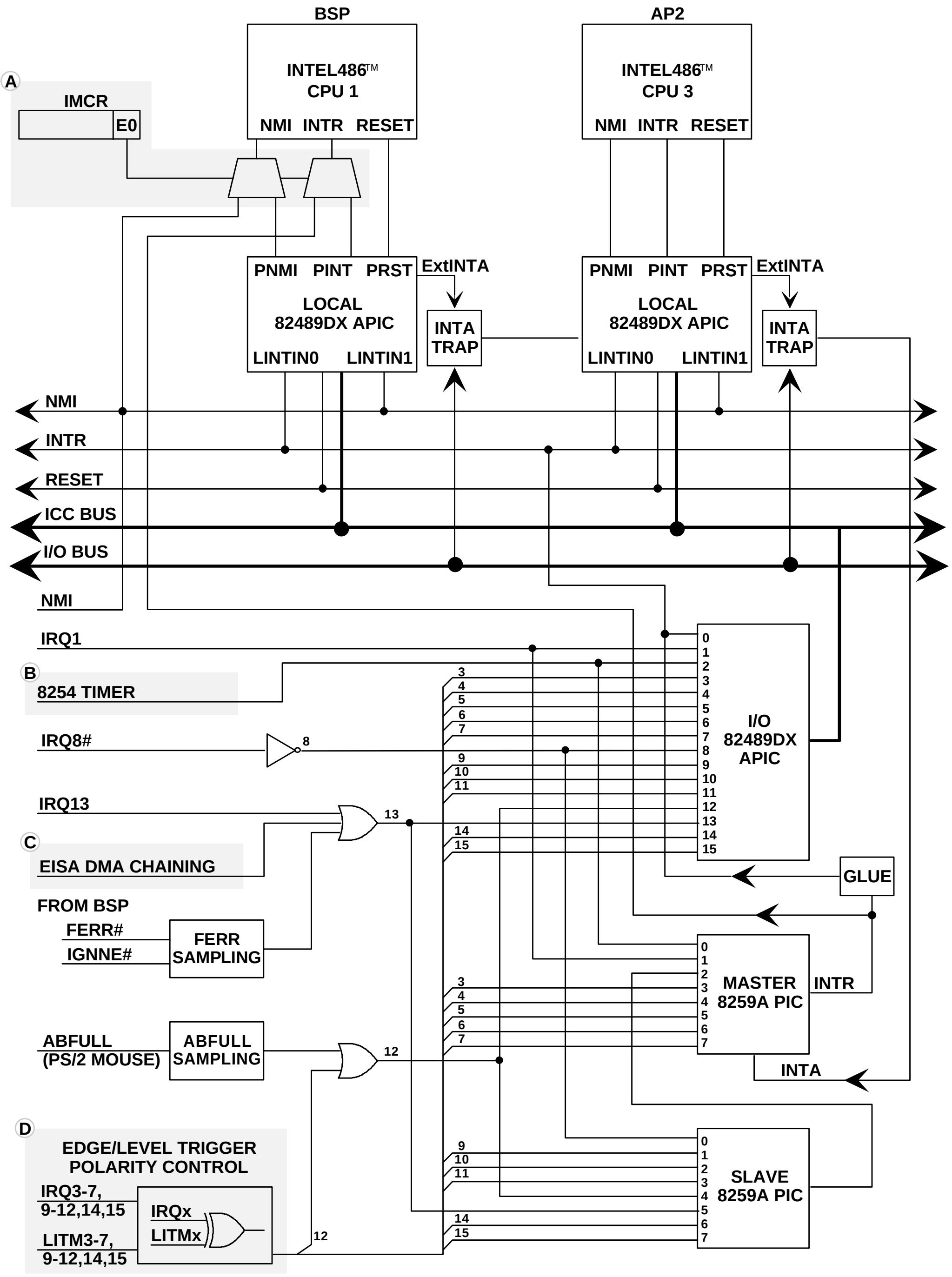
Table 5-1. Default Configurations

Default Config Code	Number of CPUs	Bus Type	APIC Type	Variant	Schematic
1	2	ISA	82489DX		As in Figure 5-1, but without EISA logic.
2	2	EISA	82489DX	Neither timer IRQ0 nor DMA chaining	As in Figure 5-1, but without IRQ0 and IRQ13 connection to the I/O APIC.
3	2	EISA	82489DX		As in Figure 5-1.
4	2	MCA	82489DX		As in Figure 5-1, but without EISA bus logic, with inverters before I/O APIC inputs 1-15.
5	2	ISA + PCI	Integrated		As in Figure 5-2, but without EISA logic.
6	2	EISA + PCI	Integrated		As in Figure 5-2.
7	2	MCA + PCI	Integrated		As in Figure 5-2, but without EISA bus logic, with inverters before I/O APIC inputs 1-15.
8-255	Reserved for MP future use.				

The default system configurations are designed to support dual-processor systems with fixed configurations. Systems with dynamically configurable components, for example, a uniprocessor system with an upgrade socket for the second processor, must always generate the MP configuration table. Failure to do so may cause the operating system to install the wrong modules due to erroneous configuration information.

5.1 Discrete APIC Configurations

Figure 5-1 shows the default configuration for systems that use the discrete 82489 APIC. The Intel486 processor is shown as an example; however, this configuration can also employ Pentium processors. In Pentium processor systems, PRST is connected to INIT instead of to RESET.



SHADED AREAS:

A: OPTIONAL IF VIRTUAL WIRE MODE IS IMPLEMENTED

B,C: MAY NOT BE EXTERNALIZED WITH SOME EISA CHIPSETS

C,D: EISA BUS SPECIFIC

Figure 5-1. Default Configuration for Discrete APIC

The INTA TRAP and GLUE in the figure are the additional hardware interface logic needed for the 82489DX APIC. INTA TRAP conditions all interrupt acknowledge cycles with ExtINTA to steer the vector either from the 8259A PIC or the APIC. INTA TRAP is also responsible for preventing the interrupt acknowledge cycle from reaching the 8259A PIC, in case ExtINTA is negated when PINT is activated. During an interrupt acknowledge cycle with ExtINTA active, the APIC does not return RDY#. Therefore, the ready generation logic should also take into consideration the status of ExtINTA to steer the ready signal either from the APIC or from external bus logic, depending upon the source of the interrupt vector.

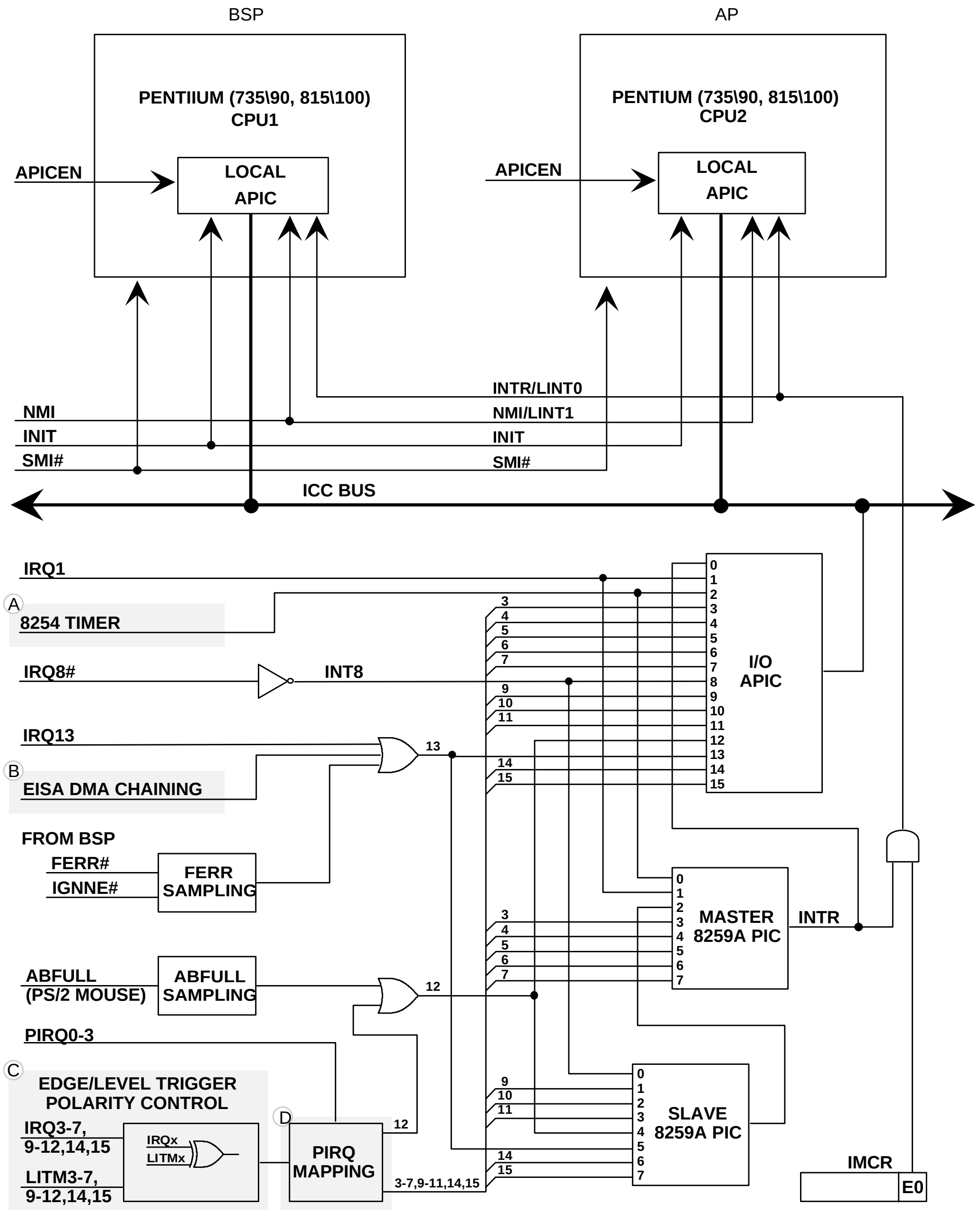
The GLUE logic converts the INTR level-triggered interrupt output signal to a signal that is acceptable to edge-triggered input pins, such as INTIN0 or LINTIN0.

If the AP used in these configurations does not automatically HALT after RESET or INIT, the AP must be prevented from executing the BIOS by external hardware or by the BIOS itself.

5.2 Integrated APIC Configurations

Figure 5-2 shows the default configuration for systems that use processors with the integrated APIC, such as Pentium processors (735\90 or 815\100). The local APIC is configured as part of the processor unit. The APICEN input is used to enable or disable the internal local APIC. If the internal local APIC is used, the BIOS must initialize the APIC to Virtual Wire Mode during system initialization.

Both Pentium processors used in the dual-processor (DP) system design are identical. The AP functions exactly the same as the BSP processor, except that the AP will go to HALT after the assertion of the RESET or INIT signals. It will remain halted until the MP operating system sends a STARTUP IPI to bring it on line.



SHADED AREAS:

A,B: MAY NOT BE EXTERNALIZED WITH SOME EISA CHIPSETS

B,C: EISA BUS SPECIFIC

D: PCI BUS SPECIFIC

Figure 5-2. Default Configuration for Integrated APIC

Two local interrupt input pins, LINT0 and LINT1, are shared with the INTR and NMI pins, respectively. The LINT0, LINT1, SMI# and INIT signals are switched by APICEN, and they should be cross-connected between the BSP and AP processors. Although the INIT pin is cross-connected between BSP and AP, a targeted INIT IPI initializes only the targeted processor, because the INIT IPI does not cause the INIT pin to change state.

The interconnection of I/O APIC interrupt lines is the same as for the 82489DX APIC configuration. However, for PCI system implementations based on the Intel PCI chipset, the PCI PIRQx lines are mapped to the ISA IRQx via a mapping register. This type of implementation makes PCI interrupt lines appear as ISA interrupt lines, which are transparent to the operating system. All PCI systems defined in the default configurations are of this type. No I/O interrupt assignment entries are declared for PCI interrupts, as described in Section 4.3.4.

5.3 Assignment of I/O Interrupts to the APIC I/O Unit

The typical APIC I/O unit has 16 general-purpose interrupt inputs. Table 5-2 shows how the interrupt request line (IRQ) assignments are connected to the I/O APIC in each of the default configurations.

Table 5-2. Default Configuration Interrupt Assignments

First I/O APIC INTINx	Config 1	Config 2	Config 3	Config 4	Config 5	Config 6	Config 7	Comments
INTIN0	8259A INTR	8259A INTR	8259A INTR	8259A INTR	8259A INTR	8259A INTR	N/C	INTR output from master 8259A or equivalent
INTIN1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	Keyboard controller buffer full
INTIN2	IRQ0	N/C	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	8254 Timer
INTIN3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	
INTIN4	IRQ4	IRQ4	IRQ4	IRQ4	IRQ4	IRQ4	IRQ4	
INTIN5	IRQ5	IRQ5	IRQ5	IRQ5	IRQ5	IRQ5	IRQ5	
INTIN6	IRQ6	IRQ6	IRQ6	IRQ6	IRQ6	IRQ6	IRQ6	
INTIN7	IRQ7	IRQ7	IRQ7	IRQ7	IRQ7	IRQ7	IRQ7	
INTIN8	IRQ8	IRQ8	IRQ8	IRQ8	IRQ8	IRQ8	IRQ8	Real time clock
INTIN9	IRQ9	IRQ9	IRQ9	IRQ9	IRQ9	IRQ9	IRQ9	
INTIN10	IRQ10	IRQ10	IRQ10	IRQ10	IRQ10	IRQ10	IRQ10	
INTIN11	IRQ11	IRQ11	IRQ11	IRQ11	IRQ11	IRQ11	IRQ11	
INTIN12	IRQ12	IRQ12	IRQ12	IRQ12	IRQ12	IRQ12	IRQ12	
INTIN13	IRQ13	N/C	IRQ13	IRQ13	IRQ13	IRQ13	IRQ13	Floating point exception and DMA chaining
INTIN14	IRQ14	IRQ14	IRQ14	IRQ14	IRQ14	IRQ14	IRQ14	
INTIN15	IRQ15	IRQ15	IRQ15	IRQ15	IRQ15	IRQ15	IRQ15	

NOTE:
N/C designates not connected.

Certain EISA chipsets do not bring out the IRQ0, 8254 timer interrupt, and IRQ13 EISA DMA chaining interrupt signals. If these signals are not directly available, INTIN2 and INTIN13 should be disabled. Refer to Section 5.3.1 for more details.

5.3.1 EISA and IRQ13

IRQ13 is a shared interrupt as defined in the EISA bus specification. Because a compliant system supports only the on-chip floating point unit, IRQ13 carries only the EISA chaining interrupt.

If IRQ13 is not connected to the I/O APIC, the EISA chaining interrupt may be handled as a mixed-mode operation. Mixed mode means that the APIC and 8259A-equivalent PIC are connected in a cascading manner via INTIN0, and INTIN0 is programmed for ExtINT and edge-triggered mode. If all other interrupts are masked off in the PIC, INTIN0 only receives the DMA chaining interrupt.

An MP operating system should disable the I/O APIC INTIN13 and configure the I/O APIC to mixed mode if the EISA DMA chaining signal is not available at the I/O APIC.

5.3.2 Level-triggered Interrupt Support

Several AT-compatible buses, such as EISA and MCA, support active-low, level-triggered interrupts. If these types of buses are to be incorporated in a compliant system, external inverters must be implemented to ensure that signals presented to the 82489DX APIC are active-high and level-triggered. See Section 4.3.4 on I/O Interrupt Assignment Flags.

For EISA implementations, the external interrupt polarity control inverters must be controlled by the EISA edge/level-triggered polarity control registers (4D0h-4D1h). MCA does not have this register. To convert an active-high trigger to an active-low trigger, an inverter for each interrupt line must be implemented.

5.4 Assignment of System Interrupts to the APIC Local Unit

The APIC local unit has two general-purpose interrupt inputs that are reserved for system interrupts. Table 5-3 shows how the interrupt request line (IRQ) assignments are connected to the local APIC in each of the default configurations.

Table 5-3 Assignment of System Interrupts to APIC Local Unit

All Local APICs LINTINx	Config 1	Config 2	Config 3	Config 4	Config 5	Config 6	Config 7	Comments
LINTIN0	8259A INTR	8259A INTR	8259A INTR	8259A INTR	8259A INTR	8259A INTR	8259A INTR	INTR output from master 8259A or equivalent
LINTIN1	NMI	NMI	NMI	NMI	NMI	NMI	NMI	Nonmaskable interrupt

The 8259A INTR output signal is connected to the LINTIN0 of all local APICs, which makes INTR dynamically routable via software. NMI is connected to the LINTIN1 of all local APICs, which makes NMI dynamically routable via software.

In PIC-Mode configurations, the NMI signal is delivered to the local interrupt input 1 (LINTIN1) of all local APICs and the input of a 2-to-1 MUX. When the system is operated in PIC Mode, the NMI is sent to the BSP directly via the MUX. The BIOS and the operating system must leave the LINTIN1 of all local APICs disabled to ensure that the BSP is the only processor that receives the NMI.

In PIC-Mode configurations, the 8259 INTR signal also follows the same convention, connecting to the local interrupt line 0 (LINTIN0) of all local APICs and the input of the second 2-to-1 MUX. When the system is operated in PIC Mode, the 8259 INTR is sent to the BSP directly via the MUX. The BIOS and the operating system must leave the LINTIN0 of all local APICs disabled to ensure that the BSP is the only processor that receives the 8259 INTR signal.

When the system is operated in Symmetric I/O Mode, the operating system may enable the LINTIN0 and LINTIN1 of any or all local APICs as necessary.

A
System BIOS Programming Guidelines

Depending on the MP components in a multiprocessor system, the system BIOS may have the following additional responsibilities:

1. Put the APs to sleep, so that they do not all try to execute the same BIOS code as the BSP. This is necessary, because BIOS code is not typically multithreaded for multiprocessing.
2. Initialize the APICs and other MP components (if any).
3. Build the MP configuration table to communicate information to the operating system about the APICs and APs.

Note that the above activities can be implemented by the hardware. The BIOS is not required to perform these activities if the hardware makes them unnecessary. For example, the system BIOS for one of the default configurations defined in Chapter 5 needs to ensure only that the MP feature information bytes identify the configuration. In all other respects, the BIOS can be the same as a standard PC/AT BIOS.

Support for the shutdown status byte (0Fh) of the PC/AT CMOS RAM is required. The startup of APs by the operating system depends on the jump to warm reset vector (40:67h) capability, as defined by the shutdown status byte. Appendix B explains this in more detail.

A.1 BIOS Post Initialization

Once system power is applied or the reset button is pressed (if the system is so equipped), a hardware circuit generates a system RESET sequence to put all the system hardware into an initial state. All active processors start to execute instructions and enter the POST (power-on self test) procedure of the BIOS, which is responsible for initializing all components in a system to a known state and for constructing various system tables in the BIOS data area (400h-4FFh) for the operating system to use.

For compliant systems that match one of the default configurations listed in Chapter 5, the work performed by the BIOS POST is minimal. The MP feature information bytes must identify the default configuration type and determine whether PIC Mode or Virtual Wire Mode is implemented.

During the system INIT or soft reset cycle, both local and I/O APICs must be reinitialized by the INIT signal and by the BIOS. This is required because the operating system will always assume that all components in the system are initialized to a known state. For the APIC, this means that all APIC registers are cleared and the local APIC ID register is initialized by the BIOS or the hardware.

Upon warm reset, the BIOS must initialize all APICs to the power on state if the warm reset signal does not physically reset the APICs.

A.2 Controlling the Application Processors

Provision must be made to prevent all processors from executing the BIOS after a power-on RESET. System developers may choose to do this by the hardware alone or by cooperation between hardware and the BIOS. In the latter case, the BIOS may be used for selecting the BSP and placing all APs to sleep after POST. The BIOS may use the APIC ID as a means by which to identify each processor and select the proper code sequence to execute. Only the selected BSP continues to load the operating system after the POST routine.

A.3 Programming the APIC for Virtual Wire Mode

The APICs do not require BIOS programming if the default interrupt mode at start-up is PIC Mode. Special programming is needed only if the startup interrupt mode is Virtual Wire Mode.

Because Virtual Wire Mode must run all existing uniprocessor software, the system BIOS must initialize and enable the BSP's APIC first. The local unit must be programmed to function as a "virtual wire," which delivers the CPU interrupt from the 8259A-equivalent PIC to the BSP via its local APIC.

The External Interrupt (ExtINT) delivery mode must be used so that the APICs and 8259A-equivalent PICs can function together in the same system. For interrupts that are programmed for ExtINT delivery mode, there is no need to issue an EOI to the APIC; only the 8259A PIC requires an EOI as usual. Also, because the 8259A delivers the vector to the processor for ExtINT delivery mode, the interrupt vector in the APIC's redirection table is ignored.

To program the APIC to Virtual Wire Mode, the system BIOS must program the APIC to enable the LINT0 of the BSP's local APIC for edge-triggered ExtINT delivery mode, and LINT1 for level-triggered NMI delivery mode. There is no need to program the I/O APIC if it is not used in Virtual Wire Mode.

Example A-1 is an example of programming the LINTIN0 and LINTIN1 to support Virtual Wire Mode.

```
;-----------------------------------------------------------------------;
; InitLocalAPIC( )                                                      ;
;-----------------------------------------------------------------------;
;                                                                       ;
;           Initialize the local APIC to virtual wire mode.             ;
;                                                                       ;
;-----------------------------------------------------------------------;
SVR             equ     0FEE000F0H
LVT1            equ     0FEE00350H
LVT2            equ     0FEE00360H

APIC_ENABLED    equ     000000100H

        public  InitLocalAPIC
InitLocalAPIC   proc    near

        push    ds                      ; save regs used for APIC init
        push    es
        push    esi

        mov     al,080h                 ; ensure NMI disabled
        out     070h,al

        in      al,021h                 ; read primary imr
        push    ax                      ; save settings
        mov     al,0ffh                 ; mask all off
        out     021h,al
        in      al,0a1h                 ; read secondary imr
        push    ax                      ; save settings
        mov     al,0ffh                 ; mask all off
        out     0a1h,al

        extrn   pmode_on : near
        call    pmode_on                ; switch into real big mode
;
; The APIC spurious interrupt must point to a vector whose lower
; nibble is 0F, that is 0xF, where x is 0 - F. Here we use Int 00FH,
; which handles spurious interrupts and supplies the necessary IRET.
; This vector is assumed to have already been initialized in memory.
;
; Enable the APIC via SVR and set the spurious interrupt to use Int 00F
;
        mov     esi,SVR
        mov     eax,[esi]               ; read SVR
        and     eax,0FFFFFF0FH          ; clear spurious vector (use vector
                                          00FH)
        or      eax,APIC_ENABLED        ; bit 8 = 1
        mov     [esi],eax               ; write SVR

;
; Program LVT1 as ExtInt, which delivers the signal to the INTR signal of all
; processors' cores listed in the destination as an interrupt that originated
; in an externally-connected interrupt controller.
;
```

Example A-1. Programming Local APIC for Virtual Wire Mode

```
        mov     esi,LVT1
        mov     eax,[esi]               ; read LVT1
        and     eax,0FFFE00FFH          ; not masked, edge, active high
        or      eax,000005700H          ; ExtInt
        mov     [esi],eax               ; write LVT1

;
; Program LVT2 as NMI, which delivers the signal on the NMI signal of all
; processors' cores listed in the destination.
;
        mov     esi,LVT2
        mov     eax,[esi]               ; read LVT2
        and     eax,0FFFE00FFH          ; not masked, edge, active high
        or      eax,000005400H          ; NMI
        mov     [esi],eax               ; write LVT2

        extrn   pmode_off : near
        call    pmode_off               ; switch back to real mode

        pop     ax                      ; restore imr settings
        out     0a1h,al                 ; restore secondary imr
        pop     ax
        out     021h,al                 ; restore primary imr
                                        ; this routine leaves NMI disabled
        pop     esi                     ; restore regs used in APIC init
        pop     es                      ; (unless also saved for CPUID)
        pop     ds
        ret

InitLocalAPIC   endp
```

Example A-1. Programming Local APIC for Virtual Wire Mode (continued)

A.4 Constructing the MP Configuration Table

For a compliant system, one of the main functions of the system BIOS is to construct the MP floating pointer structure and the MP configuration table. Because the MP configuration table is optional, the BIOS must set the MP feature information bytes in the MP floating pointer structure to indicate whether an MP configuration table is present.

If the MP configuration table is required, the BIOS constructs it in conjunction with the BSP and APs. The BIOS is responsible for synchronizing the activities of the APs during the construction of the table. The BIOS may need some synchronization during processor initialization so that each processor may be brought up in the proper order. The mechanism for synchronization is not specified; however, the procedure described in the following paragraphs of this section uses AP status flags as an example of a synchronization mechanism. This procedure also initializes the APs serially. System developers may employ other mechanisms and may initialize all processors in parallel to minimize the system start-up time.

The BIOS maintains an initialized AP status flag for each AP. Each AP will begin executing the same BIOS code as the BSP, but will eventually be put in a HALT state or held in a loop until the BSP enables its AP status flag.

The BSP is responsible for positioning the MP configuration table. The table can be located within any unreported, hidden system memory space or within the BIOS ROM region. The BIOS can select any unused space in those regions. For example, some PC/AT systems implement the Extended BIOS Data Segment, a 1-Kbyte block usually positioned at the top of the PC's 640K base memory. The configuration table may be placed in this portion of the memory map if enough free space is available.

The BIOS initializes the floating table pointer with the configuration table's memory address. Then the BSP constructs the MP configuration table based on the total number of processors, buses, I/O APICs, and interrupt sources in the system. It sets all AP processor entry bits to zero.

Next, the BSP enables each AP in turn by setting its AP status flag. Each AP follows these steps:

1. It executes a CPU identification procedure, reads its local APIC ID from the Local Unit ID Register, and uses this information to complete its entry in the MP configuration table.
2. Just prior to exiting the BIOS, the AP clears its status flag to signal the BSP to enable the next AP initialization process.
3. The AP, which is in Real Mode with interrupts disabled, executes a HLT instruction, and enters the HALT state.

After the APs complete the initialization process, the BSP continues filling in the rest of the MP configuration table entries. For I/O interrupt assignment entries, each line should have its trigger mode and polarity configured according to the device installed.

Once the MP table configuration process is complete, the BSP must calculate the checksum for the MP configuration table. The MP configuration table should be fully populated at this point. A disabled processor entry indicates that an AP failed in the initialization self-test. System developers may choose either to suspend system start up or to continue with the partially functional system.

The BSP is the only processor that is responsible for loading the operating system. It is up to the MP operating system to decide when to bring the APs on-line.

B
Operating System Programming Guidelines

The goal of the MP specification is to transfer enough information about the hardware environment to the operating system that a single, shrink-wrapped, operating-system binary can boot-up and fully utilize a wide variety of multiprocessor systems. The following sections explain how the operating system can take advantage of this specification to handle these operations:

1. Operating-system boot-up.
2. Self configuration.
3. Interrupt mode initialization.
4. Application processor startup.
5. Application processor shutdown.
6. Dynamic interrupt masking.
7. Support for unequal processors.

B.1 Operating System Boot-up

While all processors in an MP-compliant system are functionally identical, one of the processors will be designated as the boot processor (BSP) at system initialization by the system hardware or by the system hardware in conjunction with the BIOS. The rest of the processors are designated as the application processors (APs). The BSP is responsible for booting the operating system. Once the MP operating system is up and running, the BSP functions as an AP.

Usually a processor is designated as the BSP because it is capable of controlling all system hardware, including AP startup and shutdown. The operating system must determine and remember the APIC ID of the designated BSP, so it can keep the BSP operating as the last running processor during system shutdown. The BSP is not necessarily the first processor, especially in fault-tolerant MP systems in which any available processor can be designated as the BSP.

At the time that the first instruction of the operating system is executed, the APs are in the following state:

- The APs have been restrained (either by the BIOS or by the hardware) from executing operating system code.
- The APs are in a halted condition with interrupts disabled. This means that the AP's local APICs are passively monitoring the APIC bus and will react only to INIT or STARTUP interprocessor interrupts (IPIs).

The operating system's first task is to determine whether the system conforms to the MP specification. This is done by searching for the MP floating pointer structure. If a valid floating pointer structure is detected, it indicates that the system is MP-compliant, and the operating system should continue to look for the MP configuration table. If the system is not MP-compliant, the operating system may attempt other means of MP system detection, if it is capable of doing so, or treat the system as a uniprocessor system.

B.2 Operating System Booting and Self-configuration

An MP configuration table is required by the MP specification, with the exception of the default system configurations defined in Chapter 5. The table should be treated as read-only by the operating system. If the MP configuration table exists, the BSP should access the processor entries in the table to configure the operating system.

The BSP should later configure the operating system based on the bus, I/O APIC, IRQs, and system interrupt assignment entries of the configuration table. Note that certain types of buses are mutually exclusive, such as EISA with MCA, or ISA with MCA. The operating system may report such errors in the configuration table, if they occur.

If the MP configuration table does not exist, the BSP configures the operating system for the default system configuration indicated by the default configuration bits of the MP feature information bytes. In this case, only two processors and one I/O APIC exist in the system; both processors have the same type and features.

The operating system contains a set of predefined internal configuration tables that represent the default configurations described in Chapter 5. For MP-compliant systems that use one of the default configurations, the operating system derives the required configuration information from the corresponding predefined table.

To wake up the AP, the BSP should use the universal algorithm defined in Section B.4.

B.3 Interrupt Mode Initialization and Handling

At the time the operating system boots, the interrupt structure is configured for DOS compatibility. The system may be running either in PIC Mode or in Virtual Wire Mode. If it is in PIC Mode, the NMI and INTR will bypass the BSP's local APIC when the Interrupt Mode Configuration Register (IMCR) has a value of zero. The operating system should not try to read the IMCR because it may not exist.

The operating system should switch over to Symmetric I/O Mode to start multiprocessor operation. If the IMCRP bit of the MP feature information bytes is set, the operating system must set the IMCR to APIC mode. The operating system should not write to the IMCR unless the IMCRP bit is set.

Then the operating system should enable its own local APIC, thereby allowing IPI communications with other APIC-based processors. At this time, the APs' local APICs have interrupts disabled. Interrupts must remain disabled at the APs' local APICs while the BSP is enabling the I/O APIC and bringing the system to the normal operating state. Otherwise, an I/O interrupt may be delivered to the uninitialized AP, resulting in the loss of the interrupt.

It is the responsibility of the operating system to assign unique IDs to I/O APIC units.

B.4 Application Processor Startup

An AP may be started either by the BSP or by another active AP. The operating system causes application processors to start executing their initial tasks in the operating system code by using the following universal algorithm. The algorithm detailed below consists of a sequence of interprocessor interrupts and short programmatic delays to allow the APs to respond to the wakeup commands. The algorithm shown here in pseudo-code assumes that the BSP is starting an AP for documentation convenience. The BSP must initialize BIOS shutdown code to 0AH and the warm reset vector (DWORD based at 40:67) to point to the AP startup code prior to executing the following sequence:

```
BSP sends AP an INIT IPI
BSP DELAYs (10mSec)
If (APIC VERSION is not an 82489DX) {
    BSP sends AP a STARTUP IPI
    BSP DELAYs (200µSEC)
    BSP sends AP a STARTUP IPI
    BSP DELAYs (200µSEC)
}
BSP verifies synchronization with executing AP
```

Example B-1. Universal Start-up Algorithm

If the MP configuration table exists, it provides the IDs of the application processor local APICs. These IDs should be used as the destination addresses in targeted IPIs.

If the MP configuration table does not exist on an MP-compliant system, the system must be of default configuration type. The MP specification requires local APIC IDs to be numbered sequentially, starting at zero for all default configurations. As a result, the BSP can determine the AP's local APIC ID in default, two-processor configurations by reading its own local APIC ID. Since there are only two possible local APIC IDs in this case, zero and one, when the APIC ID of the BSP is one, the APIC ID of the AP is zero, and vice versa. This is important, because a BSP cannot start up an AP unless it already knows the local APIC ID.

Both INIT IPI and STARTUP IPI are open-ended commands. The operating system is responsible for determining whether its local APIC unit successfully dispatches one of these commands. The operating system must do so for an INIT and STARTUP IPI because the APIC either does not automatically retry or guarantee delivery for one of these special messages.

A local APIC unit indicates successful dispatch of an IPI by resetting the Delivery Status bit in the Interrupt Command Register (ICR). The operating system polls the delivery status bit after sending an INIT or STARTUP IPI until the command has been dispatched.

A period of 20 microseconds should be sufficient for IPI dispatch to complete under normal operating conditions. If the IPI is not successfully dispatched, the operating system can abort the command. Alternatively, the operating system can retry the IPI by writing the lower 32-bit double word of the ICR. This "time-out" mechanism can be implemented through an external interrupt, if interrupts are enabled on the processor, or through execution of an instruction or time-stamp counter spin loop.

Optionally, the operating system can make use of the information provided by the integrated APIC error register. Integrated local APIC units on Intel Architecture processors provide an APIC error register that indicates the reason for non-delivery of an APIC message. Reasons for non-delivery include SEND ACCEPT and RECEIVE ACCEPT errors that are generated when no processor responds to a message on the APIC bus. An example of this type of failure includes sending a STARTUP IPI to a destination APIC ID for a non-existent processor. Other errors are usually indicative of hardware problems and include SEND and RECEIVE CHECKSUM errors.

The operating system is responsible for determining whether the IPI was received by the targeted AP and executed successfully. For example, the operating system can define a status flag for each processor. After being awakened, a processor sets its status flag, indicating to the operating system that it is present and running. If the status flag does not change value after a certain time, the operating system should treat the processor as not present or not functional.

The following two sections describe specifics relating to the use of INIT and STARTUP IPIs that can be used to guide the implementation of the universal algorithm presented here.

B.4.1 USING INIT IPI

INIT IPIs can be used with systems based on the 82489DX APIC, or on systems that are based on multiple Pentium (735/90, 815/100) processors. INIT IPI is an Interprocessor Interrupt with trigger mode set to level and delivery mode set to "101" (bits 8 to 10 of the ICR). INIT IPIs should always be programmed as level triggered; the operating system must perform two writes to the ICR to assert and then deassert this delivery mode.

An INIT IPI is an IPI that has its delivery mode set to RESET. Upon receiving an INIT IPI, a local APIC causes an INIT at its processor. The processor resets its state, except that caches, floating point unit, and write buffers are not cleared. Then the processor starts executing from a fixed location, which is the reset vector location. To cause the processor to jump to a different location, the INIT IPI must be used as part of a warm-reset.

The warm reset is a feature of every standard PC/AT BIOS. It allows the INIT signal to be asserted without actually causing the processor to run through its entire BIOS initialization procedure (POST). This feature is used, for example, to return an 80286 processor to Real Mode. The key components of warm reset are the following:

- Shutdown code. One of the first actions of the BIOS POST procedure is to read the shutdown code from location 0Fh of the CMOS RAM. This code can have any of several values that indicate the reason that an INIT was performed. A value of 0Ah indicates a warm reset.
- Warm-reset vector. When POST finds a shutdown code of 0Ah, it executes an indirect jump via the warm-reset vector, which is a doubleword pointer in system RAM location 40:67h.

By putting an appropriate pointer in the warm-reset vector, setting the shutdown code to 0Ah, then causing an INIT, the BIOS (or the operating system) can cause the current processor to jump immediately to any location. Because all processors in an MP system share the same system memory, and because the INIT IPI gives one processor the power to cause an INIT at another, the operating system can cause any processor to jump immediately to any location.

B.4.2 USING STARTUP IPI

STARTUP IPIs are used with systems based on Intel processors with local APIC versions of 1.x or higher. These local APICs recognize the STARTUP IPI, which is an APIC Interprocessor Interrupt with trigger mode set to edge and delivery mode set to “110” (bits 8 through 10 of the ICR).

The STARTUP IPI causes the target processor to start executing in Real Mode from address 000VV000h, where VV is an 8-bit vector that is part of the IPI message. Startup vectors are limited to a 4-kilobyte page boundary in the first megabyte of the address space. Vectors A0-BF are reserved; do not use vectors in this range. STARTUP IPIs are not maskable, do not cause any change of state in the target processor (except for the change to the instruction pointer), and can be issued only one time after RESET or after an INIT IPI reception or pin assertion. A STARTUP IPI neither requires the targeted APIC to be enabled nor the interrupt table to be programmed. If the target processor is in the halted state immediately after RESET or INIT, a STARTUP IPI causes it to leave that state and start executing. The effect is to set CS:IP to VV00:0000h.

For an operating system to use a STARTUP IPI to wake up an AP, the address of the AP initialization routine (or of a branch to that routine) must be in the form of 000VV000h. Sending a STARTUP IPI with VV as its vector causes the AP to jump immediately to and begin executing the operating system’s AP initialization routine.

The operating system should not issue a STARTUP IPI to an 82489DX since the STARTUP IPI will be ignored instead of forcing the targeted processor to execute from the given address.

B.5 AP Shutdown Handling

An AP may be shut down by itself, by the BSP, or by another active AP. Shutting down an AP with an active task or a bound device driver is not permitted. Only the BSP may shut itself down, and it must be the last processor to shut down.

There are several possible alternatives for shutdown handling. Two of the possibilities are presented here by way of example.

In the simplest case, the operating system can define an IPI for shutting down and taking APs off-line. The exact usage and behavior of an IPI that is defined for this purpose is operating system specific and is not defined in this specification.

Should the BSP need to take an AP off-line or to place the AP back into a HALT state, the BSP can also use the INIT IPI with Warm Restart. The operating system places the address of a HLT instruction in the warm-reset vector (40:67), sets the CMOS shut-down code to 0Ah, then sends an INIT IPI to the AP. The INIT IPI causes the AP to enter the BIOS POST routine, where it immediately jumps to the warm-reset vector and executes the operating system’s HLT instruction. Only one processor can execute the shutdown routine at any given time, due to the use of the shutdown code. The operating system must not rely on any code being executed after the delivery of an INIT IPI used to shut down an AP. As a result, the operating system must ensure that any required state information is captured and that caches are flushed as necessary before sending the INIT IPI.

In order to do a complete system shutdown, followed by a warm restart if necessary, the operating system should return the system to a state similar to that at power-on. This includes disabling the Local APIC interrupts (LINT0/LINT1/Local APIC Timer/Error interrupt) on all processors, disabling the Local APIC on all APs and disabling all interrupts at all the I/O APICs in the system. The operating system can use an IPI or an NMI to signal to all APs for per-processor shutdown handling. The operating system may then set the CMOS shutdown code to 0Ah and perform a keyboard controller reset.

B.6 Other IPI Applications

The operating system may use IPIs for other run-time duties, such as handling the various processor caches.

B.6.1 Handling Cache Flush

The MP specification requires that hardware maintain cache coherency. Cache flushing by the operating system should not be required under normal circumstances. The only need for cache flushing by the operating system is prior to powering down a processor.

Should a system-wide cache flush be necessary, the operating system should use the broadcast IPI mechanism to request that each of the processors write back and invalidate its own cache subsystem and then synchronize upon the completion of that activity.

B.6.2 Handling TLB Invalidation

The operating system should use the IPI mechanism to request that each of the processors invalidate its TLBs. The operating system may use a broadcast IPI for this purpose. The BSP and APs should synchronize the completion of their actions either via memory-based semaphores or via targeted return IPIs. The actual IPI vector is operating system dependent.

B.6.3 Handling PTE Invalidation

The operating system should use the IPI mechanism to request that each processor invalidate a specific page-table entry (PTE) if it is cached in that processor's TLBs. The operating system may use a broadcast IPI for this purpose. The BSP and APs should synchronize the completion of their actions either via memory-based semaphores or via targeted return IPIs. The actual IPI vector is operating system dependent.

B.7 Spurious APIC Interrupts

For the 8259, there is a time window in which a spurious interrupt may be misinterpreted as a genuine interrupt. For example, if an interrupt goes inactive just after the first INTA cycle but before the second INTA cycle, the 8259 will also signal this spurious interrupt as a genuine interrupt. The distributed APIC architecture, by its nature, is more vulnerable to spurious interrupt, because the device interrupt may be latched and recognized without the INTA cycle.

To ensure that spurious interrupts are handled properly, it is strongly recommended that the device drivers must read the status register before servicing the device. In cases where spurious interrupts do occur, the device drivers may simply ignore them.

Note that this spurious interrupt is not related in any way to the Local APIC Spurious Interrupt generated by the Local APIC Error Register (0xFEE00370)

B.8 Supporting Unequal Processors

Some MP operating systems that exist today do not support processors of different types, speeds, or capabilities. However, as processor lifetimes increase and new generations of processors arrive, the potential for dissimilarity among processors increases. The MP specification addresses this potential by providing an MP configuration table to help the operating system configure itself. Operating system writers should factor in processor variations, such as processor type, family, model, and features, to arrive at a configuration that maximizes overall system performance. At a minimum, the MP operating system should remain operational and should support the common features of unequal processors.

C
System Compliance Checklist

Any "NO" answer indicates non-compliance.

Condition	YES	NO
1. PC/AT Compatibility	■	■
Does system contain all necessary MP-compatible circuitry?		
Will system boot and run DOS and Microsoft Windows?		
2. Memory Subsystem	■	■
Are system memory address map, cacheability, and shareability consistent with definitions in Table 3-1?		
Are memory-mapped I/O devices located at the top of the memory address space?		
If the system has external cache:	■	■
Is cache coherence maintained by hardware?		
Is cache flushing supported by hardware?		
Is cache flushing limited to the local caches?		
Are all locked operations visible to all processors?		
Are locked operations guaranteed on aligned memory operations?		
Are memory writes observed externally in same order as programmed?		
3. Multiprocessor Interrupt Control	■	■
Does each processor have its own local APIC?		
Is there a processor designated for booting?		
Does system support either PIC Mode or Virtual Wire Mode at power-on?		
Is Symmetric I/O Mode implemented?		
Are all APIC IDs unique?		
Are all local APIC IDs assigned by hardware or BIOS?		
Do local APIC IDs begin with zero?		
4. Reset Support	■	■
Does system implement software-initiated processor-specific INIT (INIT IPI)?		
Do system soft and hard resets reset all processors?		
5. APIC Interval Timer	■	■
On 82489DX, is CLK the only clock source of the APIC timer?		
6. Fault Resilient Booting	■	■
If fault resilient booting feature is implemented:	■	■
Is fault resilient booting feature software transparent?		
Are system signals: NMI, INTR, FERR#, IGNNE#, and A20M connected to BSP?		
7. MP Floating Pointer Structure and MP Configuration Table	■	■
Is the MP floating pointer structure implemented?		
If the MP feature byte 1 (default configuration type) is nonzero:	■	■
Does the system correctly implement one of the default configurations?		
If the MP feature byte 1 (default configuration type) is zero:	■	■
Has the system created a correct MP configuration table?		
If the IMCR presence bit is set, is PIC Mode implemented?		
If the IMCR presence bit is not set, is Virtual-Wire Mode implemented?		

D
Multiple I/O APIC
Multiple PCI Bus Systems

The information in this specification describes the majority of multiprocessor systems. This appendix provides clarifications for implementors who are considering designs with more than one I/O APIC. In particular, a number of proposed systems will incorporate multiple I/O APICs in order to support multiple PCI buses. This appendix provides guidance for implementors who wish to be sure that their designs comply with this specification.

D.1 Interrupt Routing with Multiple APICs

Two basic approaches to routing interrupts can be used when the system has more than one I/O APIC:

- The fixed routing scheme uses the same routing in both PIC or Virtual Wire Mode and symmetric I/O mode.
- The variable approach changes the routing when switching to symmetric I/O mode.

This section applies when a PCI interrupt is connected both to an I/O APIC input of its own and to the I/O APIC input of the EISA/ISA IRQ to which the interrupt is routed when in PIC or Virtual Wire Mode. This double routing is typically used to preserve PC AT compatibility at system start-up, allowing a system to boot from a disk connected to a PCI controller on a second PCI bus, for example. To prevent double delivery of this PCI interrupt once the system switches to symmetric I/O mode for an MP operating system, the duplicate routing must either be turned off or concealed from the operating system. If a PCI interrupt is only connected via an EISA/ISA IRQ, the EISA/ISA entry in the MP configuration table is sufficient to describe the routing.

The variable routing method described below is preferred since it is more flexible and offers best use of available system resources. Fixed routing is described here for compatibility with existing systems that do not implement a variable routing strategy.

D.1.1 Variable Interrupt Routing

In systems with variable interrupt routing, all PCI interrupts map to EISA/ISA IRQs when in PIC or Virtual Wire Mode. When switched to symmetric I/O mode, the system disables this routing and delivers the PCI interrupt through I/O APIC inputs different from those used by the EISA/ISA IRQs.

If IMCR is implemented, the hardware design can use this bit to enable/disable the routing of the PCI interrupts to EISA/ISA IRQs. On systems with IMCR, this operation might be the only one that is required of the operating system when switching to symmetric I/O mode, other than the actual programming of the I/O and Local APICs.

If IMCR is implemented but the system includes one or more I/O APICs that are not controlled through IMCR, the hardware must accomplish routing changes for such I/O APICs by some other means when the system switches into symmetric I/O mode. These routing changes must be done without requiring any additional intervention from software.

For systems without the IMCR register, the routing of the PCI interrupts to the EISA/ISA IRQ must be automatically disabled as the I/O APICs are programmed. Therefore, when the operating system programs the I/O APICs in accordance with the MP configuration table, the hardware must detect this operation and disable the routing mechanism without additional intervention by the operating system. This operation can be done globally for an entire system as soon as any APIC interrupts are enabled or it can be done on an interrupt-by-interrupt basis.

D.1.2 Fixed Interrupt Routing

Several implementations of fixed interrupt routing are possible, depending on hard wiring, via jumpers for example, or software means, such as chipset-specific registers. Since these implementations have no mechanism to disable the PCI interrupt to EISA/ISA IRQ routing, the MP configuration table must be set up carefully to avoid problems with duplicate interrupt mappings in symmetric I/O mode.

To avoid such problems on systems with fixed routing, PIC or Virtual Wire Mode interrupt routings must not be used by software when the system is in symmetric mode, since these routings cannot be disabled or altered. This situation implies two restrictions that must be placed on the way PCI interrupts are routed to EISA/ISA IRQs and on that way the MP configuration table is built:

- If a PCI interrupt is routed to an EISA/ISA IRQ that is used by an EISA/ISA device, that PCI interrupt must be delivered through the same I/O APIC input as that EISA/ISA IRQ. The connection from the PCI interrupt to the additional I/O APIC input must not be entered in the MP configuration table.
- If a PCI interrupt is routed to an EISA/ISA IRQ which is NOT used by an EISA/ISA device, that PCI interrupt must be delivered through its individual I/O APIC connection, and the connection of that EISA/ISA IRQ to its I/O APIC input should not be entered in the MP configuration table.

As an example, take a system with two I/O APICs, a PCI bus, and an EISA bus. Each EISA IRQ is connected to an input of one I/O APIC and each PCI interrupt is connected to the other I/O APIC. A fixed interrupt routing design could connect all PCI interrupts to a single EISA interrupt. This design does not give optimum performance, because PCI interrupts must be shareable, but it does allow all interrupts to be properly handled.

If an EISA device is connected to the same interrupt, the MP configuration table would not contain any entries for the second I/O APIC. The second I/O APIC is not used.

If no EISA device uses that interrupt, however, the MP configuration table could contain an entry for each PCI interrupt, describing its connection to the second I/O APIC. The EISA IRQ used by the PCI interrupts in PIC mode would not have an entry in the table. All other EISA IRQs would have an entry in the table describing connections to the first I/O APIC.

Fixed interrupt routing also implies a restriction on software that is implicit but important in the context of systems with more than one I/O APIC. The operating system must program I/O APICs to handle only the interrupts for which the MP configuration table contains corresponding I/O interrupt assignment entries. If the configuration table contains no entry for a given I/O APIC input, that interrupt must be left in the masked state.

D.2 Bus Entries in Systems with More Than One PCI Bus

To accommodate systems with more than one PCI bus within the confines of version 1.1 of this specification, construction of the bus entries on the MP configuration table must be handled in a very particular sequence:

1. Begin with bus entries for the PCI buses. Start at bus zero, using the actual PCI bus number as the bus ID for the bus entry.
2. Add entries for other buses. These entries can use bus ID numbers left vacant by the PCI bus entries.

This sequence implies that bus ID numbers do not have to increase sequentially by increments of one; the requirement is that they must appear in ascending order by bus ID number. This specific interpretation of the information presented in Table 4-7 ensures consistency between the information in the MP configuration table and the model for systems with multiple PCI buses that is presented in the formal PCI specification, which allows for more flexibility in bus numbering.

This numbering scheme requires bus entries in the MP configuration table to be sorted appropriately. For example, bus entries should appear in the order PCI (0), EISA (1), and PCI(4) in a system with three buses, two PCI buses numbered 0 and 4, and a single EISA bus numbered as 1.

D.3 I/O Interrupt Assignment Entries for PCI Devices

Section 4.3.4 defines the format of interrupt assignment entries. The example presented there does not, however, completely explain the semantics of the source bus IRQ field for PCI devices.

For PCI devices, the semantics for encoding PCI interrupts should mirror the PCI specification as follows:

Table D-1. I/O Interrupt Entry Source Bus IRQ Field for PCI Devices

Field	Offset (in bytes:bits)	Length (in bits)	Description
SOURCE BUS IRQ	5:0	2	Identifies the PCI interrupt signal, where 0x0 corresponds to INT_A#, 0x1 to INT_B#, 0x2 to INT_C# and 0x3 to INT_D#.
SOURCE BUS IRQ	5:2	5	Gives the PCI Device Number where the interrupt originates.
RESERVED	5:7	1	Reserved for future use.

E
Errata

The following sections provided here are intended to replace the corresponding sections in the main body of the specification. The sections provided here are shown in their entirety to provide context for the changes. Changes contained below are marked with double underlined text which shows changes relative to the version of the sections that are provided in the main body of the specification.

4.4.1 System Address Space Mapping Entries

System Address Space Mapping entries define the system addresses that are visible on a particular bus. Each bus defined in the Base Table can have any number of System Address Space Mapping entries included in the Extended Table. Thus, individual buses can be configured to support different address ranges, thereby decreasing the amount of bus traffic on a given bus and increasing the overall system performance. Each consecutive address range that is usable by the operating system to access devices on a given bus has a System Address Space Mapping entry. Figure 4-9 shows the format of each entry, and Table 4-14 explains each field.

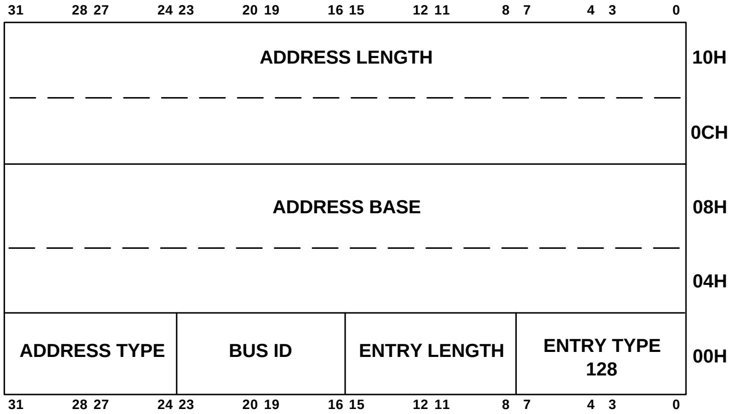


Figure 4-9. System Address Space Entry

Table 4-14. System Address Space Mapping Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	Entry type 128 identifies a System Address Space Mapping Entry.
ENTRY LENGTH	1	8	A value of 20 indicates that an entry of this type is twenty bytes long.
BUS ID	2	8	The BUS ID for the bus where the system address space is mapped. This number corresponds to the BUS ID as defined in the base table bus entry for this bus.
ADDRESS TYPE	3	8	System address type used to access bus addresses must be: 0 = I/O address 1 = Memory address 2 = Prefetch address All other numbers are reserved.
ADDRESS BASE	4	64	Starting address
LENGTH	12	64	Number of addresses which are visible to the bus

If any main memory address is mapped to a software visible bus, such as PCI, it must be explicitly declared using a System Address Space Mapping entry.

In the case of a bus that is directly connected to the main system bus, system address space records and compatibility base address modifiers must be provided as needed to fully describe the complete set of addresses that are mapped to that bus. For example, in Figure 4-10, complete explicit descriptions must be provided for PCI BUS 0 and PCI BUS 1 even if one of the buses is programmed for subtractive decode.

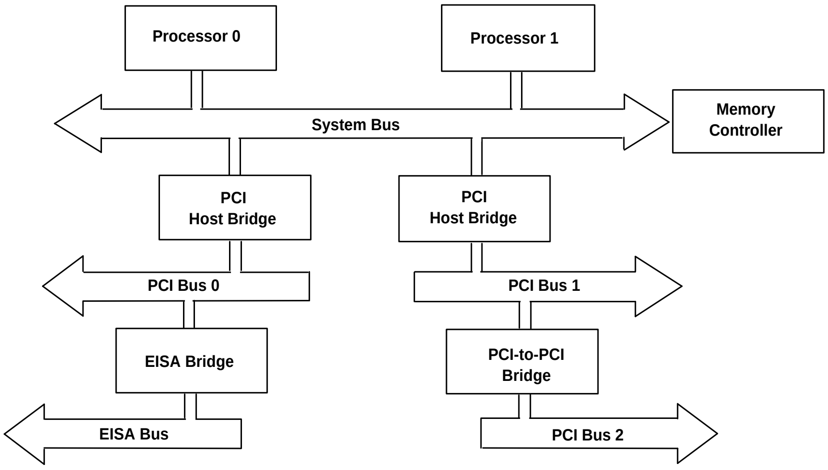


Figure 4-10. Example System with Multiple Bus Types and Bridge Types

Since all device settings must fall within supported System Address Space mapping for a given bus in order to be usable by the operating system, buses that do not support dynamically configurable devices (i.e., ISA, EISA) should support all possible addresses to that bus.

In general, the MP configuration table must provide entries to describe system address space mappings for all I/O buses present in the system. There are two exceptions to this rule:

1. For buses that are connected via PCI-to-PCI-bridge-specification-compliant bridges: in this case, the system address space mappings for such buses may be omitted from the configuration table if bus hierarchy entries for these buses are also omitted (refer to Section 4.4.2 for additional information). In such cases, the operating system is expected to discover the address space mapping by querying the PCI-to-PCI-bridge-specification-compliant bridge directly.
2. For buses that are connected via a parent I/O bus and for which the subtractive decode bit is set (refer to Section 4.4.2 for details).

Typically, this would mean that a minimal description of resources allocated to PCI buses in a system need only include System Address Space Mapping entries for PCI bus zero and any additional peer PCI buses (if present) where these buses are connected by PCI bridges that are specific to the chipset or host bus.

Note

System Address Mappings are unidirectional in nature. They only describe system addresses that propagate to the target bus from any given processor. For DMA support on the target bus, all memory addresses that contain real memory should be accessible directly by either the bus or by a bus-specific DMA controller. For buses with fewer than 32-bit address lines, all real memory at addresses that the bus can generate must be accessible for DMA.

4.4.2 Bus Hierarchy Descriptor Entry

If present, Bus Hierarchy Descriptor entries define how I/O buses are connected relative to each other in a system with more than one I/O bus. Bus Hierarchy Descriptors are used to supplement System Address Mapping entries to describe how addresses propagate to particular buses in systems where address decoding cannot be completely described by System Address Space Mapping entries alone.

In general, entries of this type are required for each bus that is connected to the system hierarchically below another I/O bus. The one exception to this is for PCI buses that are connected behind a PCI to PCI bridge specification compliant bridge. In this case the bus hierarchy entry may be omitted as the operating system is expected to discover such buses without the need to refer to the configuration table. Since bus hierarchy descriptor and system address space records are intended to work in concert, the configuration table may chose to omit both types of records for a bus behind a compliant bridge or it may provide both. The table must not however provide one without the other for buses behind compliant bridges. For example, given the system described in Figure 4-10, bus hierarchy entries are required for the EISA bus and may be provided for PCI BUS 2 since both have parent buses that are themselves I/O buses. If the configuration table does include a bus hierarchy entry for PCI BUS 2, then a complete set of corresponding system address space records must also be provided.

The Bus Hierarchy entry provides information about where in a hierarchical connection scheme a given bus is connected and the type of address decoding performed for that bus. Figure 4-11 shows the format of each entry, and Table 4-15 explains each field.

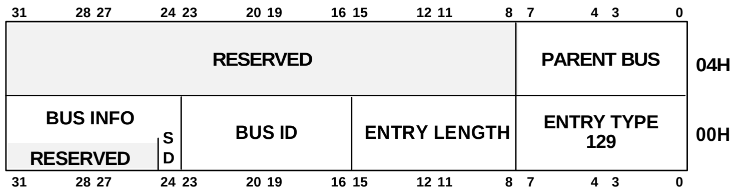


Figure 4-11. Bus Hierarchy Descriptor Entry

Table 4-15 Bus Hierarchy Descriptor Entry Fields

Field	Offset (in bytes:bits)	Length (in bits)	Description
ENTRY TYPE	0	8	Entry type 129 identifies a Bus Hierarchy Descriptor Entry.
ENTRY LENGTH	1	8	A value of 8 indicates that this entry type is eight bytes long.
BUS ID	2	8	The BUS ID identity of this bus. This number corresponds to the BUS ID as defined in the base table bus entry for this bus.
BUS INFORMATION:SD	3:0	1	Subtractive Decode Bus. If set, all addresses visible on the parent bus but not claimed by another device on the parent bus (including bridges to other buses) are useable on this bus.
PARENT BUS	4	8	Parent Bus. This number corresponds to the BUS ID as defined in the base table bus entry for the parent bus of this bus

For buses where the BUS INFORMATION:SD bit is set, System Address Mappings may not be needed. Since the bus is defined as being subtractive decode, the range of addresses that appear on the bus can be derived from address decoding information for parent and peer buses.

Glossary

82489DX: The 82489DX Advanced Programmable Interrupt Controller (APIC).

8259A: The 8259A Programmable Interrupt Controller (PIC) or its equivalent.

AP: Application processor, one of the processors not responsible for system initialization.

APIC: Advanced Programmable Interrupt Controller, either the 82489DX APIC or the integrated APIC on Pentium processors.

BIOS: Basic Input/Output Subsystem.

BSP: Bootstrap processor, the processor responsible for system initialization.

Cache coherency: A property of a cache/memory system that guarantees that a request for an item from memory will retrieve the most up-to-date value of that item, regardless of what cache or memory location currently holds that value.

CMOS RAM: The battery backed-up configuration memory of the PC/AT motherboard.

DP: A dual processor system is one with two processors.

ExtINT: A delivery mode of the Local Vector Table of a local APIC that causes delivery of a signal to the INT pin of the processor as an interrupt that originated in an externally connected 8259A-equivalent PIC. The ExtINTA output signal is also asserted. The INTA cycle that corresponds to the ExtINT delivery should be routed to the external PIC that is expected to supply the vector.

Flush: Write back all modified lines of a cache.

INIT: Unless otherwise specified, the processor-specific reset or system-wide soft reset functions. This definition is functional and sometimes bears no relationship to the actual signal name. For example, the term "INIT" may refer to the INIT signal on the Pentium processor or to the RESET signal on the Intel486 processor.

INIT IPI: A type of APIC interprocessor interrupt whose delivery mode is set to RESET. Upon delivery to the specified destinations, the destination APICs assert their PRST output signals. When the PRST lines are connected to the INIT or RESET inputs of their respective processors, an INIT IPI causes reinitialization of the destination processors.

Invalidate: Change the state of a cache line to the Invalid state.

IPI: Interprocessor interrupt.

MESI: A cache coherency protocol named after the states that cache lines may have: Modified, Exclusive, Shared, Invalid.

MP: A multiprocessor system is one with two or more processors.

PIC: Programmable Interrupt Controller.

PIC Mode: One of three interrupt modes defined by the MP specification. In this mode the APICs are effectively disabled, while interrupts are generated by 8259A-equivalent PICs and delivered directly to the BSP. This is a uniprocessor compatibility mode.

POST: Power-On Self Test, the first BIOS procedure executed after a RESET or INIT.

RESET: The system-wide hard reset. This definition is functional. It may refer to the RESET signal on both Pentium and Intel486 processors or the RESET signal of the 82489DX APIC.

Shutdown code: The value of CMOS RAM location 0Fh, which indicates that reason that a RESET was performed.

STARTUP IPI: A type of APIC interprocessor interrupt that is similar to an NMI with an embedded vector. It does not cause any change of state, but merely causes the targeted processor to start executing in Real Mode from address 000VV000h, VV being an 8-bit vector which is part of the IPI message. Startup vectors are limited to a 4K page boundary in the first 1 MB of the address space. STARTUP IPIs are not maskable, and can be issued at any time. The benefit of STARTUP-IPI compared to NMI is that it does not require the targeted APIC to be enabled, and it does not require the interrupt table to be programmed. Thus, the operating system's initialization procedure can use it to wake up an AP that has been sleeping since RESET or INIT. The STARTUP IPI is not supported by the 82489DX APIC.

Symmetric I/O Mode: One of three interrupt modes defined by the MP specification. In this mode, the APICs are fully functional, and interrupts are generated and delivered to the processors by the APICs. Any interrupt can be delivered to any processor. This is the only multiprocessor interrupt mode.

Symmetry: The relationship of equality among components of a multiprocessor system in which no processor is special with respect to its access to memory, interrupts, or I/O. For interrupts, symmetry means that any interrupt from any source can be routed to any processor and handled there. For I/O, it means that all I/O control registers, be they in memory space, I/O space, or some other special address space, are accessible to all processors. Per-processor control hardware, such as interrupt controllers or processor identification registers, must be at the same physical address for all processors.

Virtual Wire Mode: One of three interrupt modes defined by the MP specification. In this mode interrupts are generated by the 8259A-equivalent PICs, but delivered to the BSP by an APIC that is programmed to act as a "virtual wire"; that is, the APIC is logically indistinguishable from a hardwired connection. This is a uniprocessor compatibility mode.

Warm reset: A technique that allows the RESET or INIT signal to be asserted without actually causing the BIOS to run through its entire initialization procedure. If a value of 0Ah is placed in the **shutdown code**, the first instructions of the BIOS POST procedure read the warm-reset vector from system RAM location 40:67h, and jump to that address.

Write-back: A cache update policy wherein a modified cache line is not written back to main memory until the last possible instant—when another processor needs to access the data or when its location in the cache is needed for other data.

Order Number: 242016-004

Printed in U.S.A.