IBM *Power Personal Systems* Architecture

Residual Data

Document Number: PPS-AR-FW0001

July 10, 1996

Version 0.6
Next Revision Planned: As Required

Document Owner:

Raymond J. Pedersen
Dept. WK3
Zip 4357
(512) 838-6536
rj@vnet.ibm.com

Residual Data

Document Number: PPS-AR-FW0001

Document Change List

Version 0.1: (July 10,1995) Initial version

Added over previously current header files:

1. PnP_SUB_TYPE = VMEBridge = 6
2. PnP_INTERFACE = GeneralVMEBridge = 0
3. BUS_ID = VMEDEVICE = 0x100

Version 0.2: (July 24, 1995)

1. Residual.VitalProductData.ExtendedVPD changed from a *pointer* to an unsigned long *offset*.
2. EXTVPDPNP.H (IBM type = 0x10) added.
3. Comment added regarding use of DevID field (Motorola request).

Version 0.3: (October 5, 1995)

1. FirmwareSupplier = FirmWorks added.
2. PnP_SUB_TYPE = SystemPlanar and PnP_INTERFACE = GeneralSystemPlanar added
3. Grackle added to CHPIDPNP.H.

Version 0.4: (December 8, 1995)

1. IBM2782352 added to CHPIDPNP.H as IBM8109
2. TBCTLPNP.H added: Timebase Control PNP packet.

Version 0.5: (April 3, 1996)

1. RESIDUAL.H: FirmwareSupplier = Bull added (0x03).
2. PnP_INTERFACE: CHRP_Floppy and CHRPSystemStatusLED added
3. Dallas1585 added to CHPIDPNP.H.
4. 12.1” Color TFT Displays added to DISPPNP.H.

Version 0.6: (July 10, 1996)

1. RESIDUAL.H: typedef enum CPU_STATE: add CPU_NOT_PRESENT = 255

1.0 Residual Data

1.1 Introduction

Residual Data is a data structure that is generated primarily by the system firmware and placed in memory prior to the transfer of control to the operating system. Register r3 is left with a pointer to this data. The data consists of a header area followed by a structured description of the underlying hardware. A major portion of this information is a device tree that is fashioned after the ISA Plug and Play Specification; the DevicePnPHeap is a collection of Plug and Play resource tags.

This document contains definitions and explanations of all elements of the Residual Data structure for PowerPC Reference Platform systems. The C language structure files covering residual data are included as an appendix to this document but the latest levels should be obtained via anonymous FTP from ftp.austin.ibm.com in directory /pub/technology/spec.

The residual data structure consists of:

- `residual.h` contains the header and fixed fields of the structure (Appendix A).
- `pnp.h` contains enumerations associated with various components of `residual.h` and defines the structure of the available *Plug and Play* resource tags (Appendix B).
- A set of *vendor-specific Plug and Play* tags defined specifically for IBM PowerPC Reference Platform systems.

1.2 Notation

Standard C Language notation may be assumed throughout this document.

All structures use Big-Endian data format except for the *Plug and Play* resource tags (in RESIDUAL.DevicePnPHeap) which are in Little Endian format in accordance with the *Plug and Play* architecture.

Reserved fields in all structures must be set to zero by the firmware.

1.3 residual.h

This section contains explanations of the terminology and intended usage of residual data fields. The figure below is a high level view of the entire structure; details of individual fields follow. In this representation, groups of fields are marked with an asterisk to indicate that they are part of an array. These fields are typically repeated for all elements of the array, but are shown here only once for clarity and brevity. Each row in the figure is 4 bytes in length with offset 0 at the top.

Residual Data Structure

Length			
Version	Revision	EC	
VitalProductData (236 Bytes)			
MaxNumCpus		ActualNumCpus	
CpuType*			
CpuNumber*	CpuState*	Reserved*	
TotalMemory			
GoodMemory			
ActualNumMemSegs			
Usage*			
BasePage*			
PageCount*			
ActualNumMemories			
SIMMSize*			
ActualNumDevices			
BusId*			
DevId*			
SerialNum*			
Flags*			
BaseType*	SubType*	Interface*	Spare*
BusAccess*			
AllocatedOffset*			
PossibleOffset*			
CompatibleOffset*			
DevicePnPHeap (n Bytes)			

1.3.1 RESIDUAL.ResidualLength (4 Bytes)

This is the length, in bytes, of the entire residual data structure as it would reside in memory. This length is calculated as a function of MAX_CPUS, MAX_MEM_SEGS, MAX_MEMS, MAX_DEVICES, and AVE_PNP_SIZE. RESIDUAL.ResidualLength is, therefore, *fixed* for a particular *version* of the structure.

1.3.2 RESIDUAL.Version (1 Byte)

Contains the *version* of the residual data structure. The version starts with 0 and gets increased by 1 whenever a change is made that affects fields other than the field location being changed , i.e. offsets into the structure are affected.

1.3.3 RESIDUAL.Revision (1 Byte)

Contains the *revision* of the residual data structure. The revision starts with 0 and gets increased by 1 when a change is made to an existing field (other than a reserved field) that does *not* affect the binary compatibility of the other parts of the structure. Adding meaning to a reserved field or adding a new flag or decode point does *not* constitute a revision change.

1.3.4 RESIDUAL.EC (2 Bytes)

Contains the *engineering change level* of the residual data structure. The EC level starts with 0 and gets increased by 1 when a fix is made to an existing version/revision.

1.3.5 RESIDUAL.VitalProductData (236 Bytes)

Vital Product Data (VPD) is information about system components that is useful in establishing the identity and capabilities of the system. See Table 1.4 on page 12 for details of this part of the structure.

1.3.6 RESIDUAL.MaxNumCpus (2 Bytes)

Contains the maximum number of cpus that *is supported by* the system being described. This is defined differently than MAX_CPUS, which is the maximum number of cpus for this version of the structure and is a parameter of the structure size.

1.3.7 RESIDUAL.ActualNumCpus (2 Bytes)

This is the *actual* number of cpus *installed* on the system being described. If this is less than MaxNumCpus, there are unpopulated or otherwise unusable processor locations in the system.

1.3.8 RESIDUAL.Cpus[MAX_CPUS] (8 Bytes per CPU)

This is an array of elements identifying each cpu in the system. Each element contains the following fields (typedef PPC_CPU):

- RESIDUAL.Cpus[].CpuType (4 Bytes): CPU version/revision that results from a mfspr from the Processor Version Register (PVR).
- RESIDUAL.Cpus[].CpuNumber (1 Byte): This cpu number identifies a cpu in a multiprocessor configuration and corresponds to the number used for interprocessor interrupts. The CpuNumber for Cpu[0] in a uniprocessor system must be 0.
- RESIDUAL.Cpus[].CpuState (1 Byte):

Table 1. RESIDUAL.Cpus[].CpuState

Value	Name	Description
0x00	CPU_GOOD	This Cpu is present and active
0x01	CPU_GOOD_FW	This Cpu is present and executing firmware code.
0x02	CPU_OFF	This Cpu is present, but not active.
0x03	CPU_FAILED	This Cpu is present and active, but failed post.
0xFF	CPU_NOT_PRESENT	The Cpu is not present.

- RESIDUAL.Cpus[].Reserved (2 Bytes)

1.3.9 RESIDUAL.TotalMemory (4 Bytes)

The total number of bytes of system memory installed. If RESIDUAL.VitalProductData.WordWidth = 64, the field is expressed in pages (RESIDUAL.VitalProductData.PageSize) rather than bytes.

1.3.10 RESIDUAL.GoodMemory (4 Bytes)

The total number of bytes of good system memory installed. If RESIDUAL.VitalProductData.WordWidth = 64, the field is expressed in pages (RESIDUAL.VitalProductData.PageSize) rather than bytes.

1.3.11 RESIDUAL.ActualNumMemSegs (4 Bytes)

Indicates the number of elements of the RESIDUAL.Segs array that are actually used.

1.3.12 RESIDUAL.Segs[MAX_MEM_SEGS] (12 Bytes per Memory

Segment)

This is an array of elements identifying each memory segment in the system. Each element contains the following fields (typedef MEM_MAP):

- RESIDUAL.Segs[].Usage (4 Bytes): gives the intended usage of the memory segment as given in Table 2 on page 9.
- RESIDUAL.Segs[].BasePage (4 Bytes): gives the base address as a page number.
- RESIDUAL.Segs[].PageCount (4 bytes): gives the size of the segment in pages.

Table 2. RESIDUAL.Segs[].Usage

Value	Name	Description
0x0		Undefined
0x01	FirmwareStack	Stack for firmware usage
0x02	FirmwareHeap	Heap area for firmware usage
0x04	FirmwareCode	Code area for firmware usage
0x08	BootImage	Location of boot image
0x10	Free	Free memory range
0x20	Unpopulated	Unpopulated memory range
0x40	ISAAddr	ISA address space
0x80	PCIConfig	PCI configuration area
0x100	PCIAddr	PCI address space
0x200	SystemRegs	System register area
0x400	SystemIO	System IO area
0x800	IOMemory	IO memory space
0x1000	UnPopSystemROM	Unpoulated part of system ROM area
0x2000	SystemROM	System ROM area (populated)
0x4000	ResumeBlock	Power management usage
0x8000	Other	Other usage

1.3.13 RESIDUAL.ActualNumMemories (4 Bytes)

Indicates the number of elements of the RESIDUAL.Memories array that are actually used.

1.3.14 RESIDUAL.Memories[MAX_MEMS] (4 Bytes per Memory Module)

This is an array of elements identifying each physical memory module in the system. Each element contains the following field (typedef PPC_MEM):

- RESIDUAL.Memories[].SIMMSize (4 Bytes): size of the SIMM in MB. 0 means absent or bad.

1.3.15 RESIDUAL.ActualNumDevices (4 Bytes)

Indicates the number of elements of the RESIDUAL.Devices array that are actually used.

1.3.16 RESIDUAL.Devices[MAX_DEVICES] (36 Bytes per Device)

This is an array of elements identifying each I/O device in the system. Each element contains the following fields (typedef PPC_DEVICE):

- RESIDUAL.Devices[].DeviceId (20 Bytes) is a structure that identifies the I/O device. See Table 3 on page 10 for the components of this structure.

Table 3. RESIDUAL.Devices[].DeviceId

Component	Size	Description
BusId	4 bytes	Identifies the bus on which this device resides. 0x01 = ISADEVICE 0x02 = EISADEVICE 0x04 = PCIDEVICE 0x08 = PCMCIADEVICE 0x10 = PNPISADEVICE 0x20 = MCADEVICE 0x40 = MXDEVICE 0x80 = PROCESSORDEVICE 0x0100 = VMEDEVICE 0x0200-0x80000000 = undefined
DevId	4 bytes	Device Identifier appropriate for the BusId. Use of this field is intended primarily for ISA devices; most other devices are readily identified through their configuration methods. Current usage is limited to ISA devices and PCI-PCI bridges which both use PnP device ids.
SerialNum	4 bytes	Used to distinguish multiple occurrences of the same DevId

Table 3. RESIDUAL.Devices[].DeviceId

Component	Size	Description
Flags	4 bytes	Provides system characteristics associated with the device. 0x0001 = Output 0x0002 = Input 0x0004 = ConsoleOut 0x0008 = ConsoleIn 0x0010 = Removable 0x0020 = ReadOnly 0x0040 = PowerManaged 0x0080 = Disableable 0x0100 = Configurable 0x0200 = Boot - device can be used by boot. 0x0400 = Dock - device is on a docking station. 0x0800 = Static - not dynamically configurable. 0x1000 = Failed - device failed POST. 0x2000 = Integrated - device is integrated (on the planar), not removable. 0x4000 = Enabled 0x8000-0x80000000 = undefined
BaseType	1 byte	Most general classification of the device. See Table 6 on page 15
SubType	1 byte	More specific classification of the device. See Table 6 on page 15
Interface	1 byte	Characterizes the programming interface for the device. See Table 6 on page 15
Spare	1 byte	

- RESIDUAL.Devices[].BusAccess (4 Bytes) is a union of the structures shown in Table 4 on page 11

Table 4. RESIDUAL.Devices[].BusAccess

Struct name	Element name	Size	Description
PnPAccess	CSN	1 byte	Card Select Number
	LogicalDevNumber	1 byte	
	ReadDataPort	2 bytes	Relocatable within the I/O range 0x0203 to 0x03ff.
ISAAccess	SlotNumber	1 byte	0 if unknown
	LogicalDevNumber	1 byte	0 if unknown
	ISAReserved	2 bytes	0
MCAAccess	SlotNumber	1 byte	
	LogicalDevNumber	1 byte	
	MCAReserved	2 bytes	0

Table 4. RESIDUAL.Devices[].BusAccess

Struct name	Element name	Size	Description
PCMCIAAccess	SlotNumber	1 byte	
	LogicalDevNumber	1 byte	
	PCMCIAReserved	2 bytes	0
EISAAccess	SlotNumber	1 byte	0 if unknown
	FunctionNumber	1 byte	0 if unknown
	EISAReserved	2 bytes	0
PCIAccess	BusNumber	1 byte	
	DevFuncNumber	1 byte	
	PCIReserved	2 bytes	0
ProcBusAccess	BusNumber	1 byte	
	BUID	1 byte	
	ProcBusReserved	2 bytes	0

- RESIDUAL.Devices[].AllocatedOffset (4 Bytes): Offset into RESIDUAL.DevicePnPHeap for the allocated resource description.This points to the list of *Plug and Play* descriptors for resources that are actually allocated to the device.
- RESIDUAL.Devices[].PossibleOffset (4 Bytes): Offset into RESIDUAL.DevicePnPHeap for the possible resource description.This points to the list of *Plug and Play* descriptors for resources that are alternatives to be used in the event of resource contention or reconfiguration.
- RESIDUAL.Devices[].CompatibleOffset (4 Bytes): Offset into RESIDUAL.DevicePnPHeap for the compatible resource description.This points to the list of *Plug and Play* descriptors for resources that are compatible with the allocated device.

1.3.17 RESIDUAL.DevicePnPHeap[2*MAX_DEVICES*AVE_PNP_SIZE]

The DevicePnPHeap is a collection of *Plug and Play* resource descriptors. Access to the descriptor list for each device is provided through RESIDUAL.Devices[].AllocatedOffset, RESIDUAL.Devices[].PossibleOffset, and RESIDUAL.Devices[].CompatibleOffset.

1.4 Vital Product Data

Vital Product Data (VPD) is information about the hardware platform or configuration that is not easily determined directly by the software. Some amount of VPD is provided by the firmware in the residual data structure. Refer to Table 5 on page 13 for details of this data structure.

Table 5. Vital Product Data

Name	Size	Description
PrintableModel	32 bytes	This is a null terminated string that must be of the form: vvv<0x20><model designation><0x0> where vvv is the EISA Vendor ID. Example: IBM PPS MODEL 6015<trailing nulls>
Serial	16 bytes	The serial number must be of the form: vvv<serial number> where vvv is the EISA Vendor ID. Example: IBM6015123456789
Reserved	48 bytes	Must be set to 0x0 by the firmware.
FirmwareSupplier	4 bytes	IBMFirmware = 0x00 MotoFirmware = 0x01 FirmWorks = 0x02 Bull = 0x03 Firmware suppliers requiring unique identification in this field should contact rj@ausvm6.vnet.ibm.com.
FirmwareSupports	4 bytes	0x0001 = Conventional - this refers to pre- Open Firmware implementations. 0x0002 = OpenFirmware 0x0008 = LowDebug 0x0010 = Multiboot 0x0020 = LowClient 0x0040 = Hex41 - see the Reference Platform boot process for further information. 0x0080 = FAT - File Allocation Table file system supported. 0x0100 = ISO9660 - CD-ROM format supported 0x0200 = SCSI_Initiator_Override - firmware allows the SCSI initiator id to be overridden to support multi-initiator configurations. See section on NVRAM. 0x0400 = Tape_Boot supported 0x0800 = FW_Boot_Path - see NVRAM specification. 0x1000-0x80000000 = undefined
NvramSize	4 bytes	The size of NVRAM in bytes.
NumSIMMSlots	4 bytes	Number of SIMM slots in the system.
EndianSwitchMethod	2 bytes	There are currently two methods for changing the endian mode of the memory controller. 0x00 = undefined 0x01 = UsePort92 - ISA port 0x92 0x02 = UsePCIConfigA8 - PCI configuration register 0xa8, bit 5. 0x03 = UseFF001030 - bit 9 0x04-0xFFFF = undefined
SpreadIOMethod	2 bytes	There are currently two methods for setting contiguous/noncontiguous I/O mode: 0x00 = UsePort850 - ISA port 0x850 0x01 = UsePCIConfigA8 - PCI configuration register 0xa8, bit 19 0x02-0xFFFF = undefined
SmpIar	4 bytes	Symmetrical multiprocessor Instruction address register - set by the MP operating system to tell the firmware where to branch to on termination of a spin loop during MP initialization.
RAMErrorLogOffset	4 bytes	Offset into RESIDUAL.DevicePnPHeap for Error Log Descriptor.
Reserved5	4 bytes	Must be set to 0x0 by the firmware.

Table 5. Vital Product Data

Name	Size	Description
Reserved6	4 bytes	Must be set to 0x0 by the firmware.
ProcessorHz	4 bytes	Processor speed in Hertz.
ProcessorBusHz	4 bytes	Processor Bus speed in Hertz.
Reserved7	4 bytes	Must be set to 0x0 by the firmware.
TimeBaseDivisor	4 bytes	Defined as the number of bus clocks per timebase clock times 1000. For example, if the timebase is running at 1/4 the rate of the bus clock, the TimeBaseDivisor is 4000.
WordWidth	4 bytes	Set to 32 for 32-bit PowerPC processor implementations and to 64 for 64-bit implementations.
PageSize	4 bytes	Memory page size in bytes; most often set to 4096.
CoherenceBlockSize	4 bytes	Unit of the L1 cache on which coherency is maintained. This is normally the same as the CacheLineSize.
GranuleSize	4 bytes	Unit if the L1 Cache used by an operating system to allocate lock variables; normally the same as the CacheLineSize.
CacheSize	4 bytes	The size of the L1 Cache in K bytes. This is the *total* size of the cache, whether combined or split.
CacheAttrib	4 bytes	L1 Cache attributes: 0 = NoneCAC - no cache 1 = SplitCAC - Split (I&D) cache 2 = CombinedCAC - Combined (I&D) cache
CacheAssoc	4 bytes	L1 Cache Associativity. This is used for a combined cache; if the cache is split, put zeros here.
CacheLineSize	4 bytes	L1 Cache size in bytes. This is used for a combined cache; if the cache is split, put zeros here.
I_CacheSize	4 bytes	L1 Instruction Cache size in bytes. For combined cache, put total here.
I_CacheAssoc	4 bytes	L1 Instruction Cache associativity. For combined cache, put total here
I_CacheLineSize	4 bytes	L1 Instruction Cache line size in bytes. For combined cache, put total here
D_CacheSize	4 bytes	L1 Data Cache size in bytes. For combined cache, put total here.
D_CacheAssoc	4 bytes	L1 Data Cache associativity. For combined cache, put total here
D_CacheLineSize	4 bytes	L1 Data Cache line size in bytes. For combined cache, put total here
TLBSize	4 bytes	Total size of Translation Lookaside Buffer in *number of entries*.
TLBAttrib	4 bytes	Translation Lookaside Buffer attributes: 0 = NoneTLB - none 1 = SplitTLB - split (I&D) TLB 2 = CombinedTLB - combined (I&D) TLB
TLBAssoc	4 bytes	TLB Associativity. This is used for a combined TLB; if split, put zeros here.

Table 5. Vital Product Data

Name	Size	Description
I_TLBSize	4 bytes	Number of Instruction TLB entries; for combined TLB, put total here.
I_TLBAssoc	4 bytes	Instruction TLB associativity; for combined TLB, put total here.
D_TLBSize	4 bytes	Number of Data TLB entries;. for combined TLB, put total here.
D_TLBAssoc	4 bytes	Data TLB associativity; for combined TLB, put total here.
ExtendedVPD	4 bytes	Offset for extended VPD (generally into DevicePnPHeap); zeros if unused.

1.5 Device Types

Table 6. Plug and Play Device Types

PnP_BASE_TYPE	PnP_SUB_TYPE	PnP_INTERFACE
0 = Reserved		
1 = MassStorageDevice	0 = SCSIController	0 = GeneralSCSI
	1 = IDEController	0 = GeneralIDE 1 = ATACompatible
	2 = FloppyController	0 = GeneralFloppy 1 = Compatible765 2 = NS398_Floppy - NS SuperI/O with index/data regs at port 398 3 = NS26E_Floppy - NS at port 26E 4 = NS15C_Floppy - NS at port 15C 5 = NS2E_Floppy - NS at port 02E 6 = CHRP_Floppy
	3 = IPIController	0 = GeneralIPI
	0x80 = OtherMassStorageControl-ler	
2 = NetworkInterface Controller	0 = EthernetController	0 = GeneralEther
	1 = TokenRingController	0 = GeneralToken
	2 = FDDIController	0 = GeneralFDDI
	0x80 = OtherNetworkController	
3 = DisplayController	0 = VGAController	0 = GeneralVGA
	1 = SVGAController	0 = GeneralSVGA
	2 = XGAController	0 = GeneralXGA
	0x80 = OtherDisplayController	

Table 6. Plug and Play Device Types

PnP_BASE_TYPE	PnP_SUB_TYPE	PnP_INTERFACE
4 = MultimediaController	0 = VideoController	0 = GeneralVideo
	1 = AudioController	0 = GeneralAudio 1 = CS4232Audio
	0x80 = OtherMultimediaController	
5 = MemoryController	0 = RAM	0 = GeneralRAM 0 = PCIMemoryController - PCI config. 1 = RS6KMemoryController
	1 = FLASH	0 = GeneralFLASH
	0x80 = OtherMemoryDevice	
6 = BridgeController	0 = HostProcessorBridge	0 = GeneralHostBridge
	1 = ISABridge	0 = GeneralISABridge
	2 = EISABridge	0 = GeneralEISABridge
	3 = MicroChannelBridge	0 = GeneralMCABridge
	4 = PCIBridge	0 = PCIBridgeDirect - memory mapped PCI bus configuration 1 = PCIBridgeIndirect - PCI bus configuration space accessed through CF8, CFC 2 = PCIBridgeRS6K - PCI bus configuration space accessed through system control area (high 16 MB - used by RS/6000 systems).
	5 = PCMCIABridge	0 = GeneralPCMCIABridge
	6 = VMEBridge	0 = GeneralVMEBridge
	0x80 = OtherBridgeDevice	
7 = CommunicationsDevice	0 = RS232Device	0 = GeneralRS232 1 = COMx 2 = Compatible16450 3 = Compatible16550 4 = NS398SerPort - NS SuperI/O with index/data regs at port 398 5 = NS26ESerPort - NS at port 26E 6 = NS15CSerPort - NS at port 15C 7 = NS2ESerPort - NS at port 02E
	1 = ATCompatibleParallelPort	0 = GeneralParPort 1 = LPTx 2 = NS398ParPort - NS SuperI/O with index/data regs at port 398 3 = NS26EParPort - NS at port 26E 4 = NS15CParPort - NS at port 15C 5 = NS2EParPort - NS at port 02E
	0x80 = OtherCommunicationsDevice	

Table 6. Plug and Play Device Types

PnP_BASE_TYPE	PnP_SUB_TYPE	PnP_INTERFACE
8 = SystemPeripheral	0 = ProgrammableInterruptController	0 = GeneralPIC 1 = ISA_PIC - 8259 2 = EISA_PIC 3 = MPIC 4 = RS6K_PIC (See RS/6000 interrupt architecture)
	1 = DMAController	0 = GeneralDMA 1 = ISA_DMA 2 = EISA_DMA
	2 = SystemTimer	0 = GeneralTimer 1 = ISA_Timer 2 = EISATimer
	3 = RealTimeClock	0 = GeneralRTC 1 = ISA_RTC
	4 = L2Cache	0 = none 1 = StoreThruOnly 2 = StoreInEnabled 3 = RS6KL2Cache
	5 = NVRAM	0 = IndirectNVRAM - indirectly addressed through ports 74-76 1 = DirectNVRAM - direct memory mapped NVRAM. 2 = IndirectNVRAM24 - 24 bit addressing, indirectly addressed using ports 73-76.
	6 = PowerManagement	0 = GeneralPowerManagement 1 = EPOWPowerManagement - Emergency Power Off Warning 2 = PowerControl - software control over power on/off
	7 = CMOS	0 = GeneralCMOS
	8 = OperatorPanel	0 = GeneralOPPanel 1 = HarddiskLight 2 = CDROMLight 3 = PowerLight 4 = KeyLock 5 = ANDisplay - Alphanumeric Display (example: LCD panel) 6 = SystemStatusLED 7 = CHRPSystemStatusLED
	9 = ServiceProcessorClass1	0 = GeneralServiceProcessor
	0xA = ServiceProcessorClass2	0 = GeneralServiceProcessor
	0xB = ServiceProcessorClass3	0 = GeneralServiceProcessor
	0xC = GraphicAssist	0 = none 1 = TransferData 2 = IGMC32 3 = IGMC64
	0xF = SystemPlanar	0 = GeneralSystemPlanar
	0x80 = OtherSystemPeripheral	

Table 6. Plug and Play Device Types

PnP_BASE_TYPE	PnP_SUB_TYPE	PnP_INTERFACE
9 = InputDevice	0 = KeyboardController 1 = Digitizer 2 = MouseController 0x80 = OtherInputContoller	

1.6 Plug and Play Device Descriptors (Vendor Defined)

Refer to the Plug and Play ISA Specification for a description of the tag architecture for device description. Only the tags defined by IBM will be discussed here (small resource item 0xE and large resource item 0x4). These have been defined to meet IBM's specific product needs (primarily AIX), but will probably be useful to other platforms and OSs. Vendors should create new vendor-defined tags as needed.

The primary documentation of these descriptors is the set of header files embedded in the following sections. Some clarification of potentially misinterpreted or confusing fields is included.

The term *descriptor* may be used in addition to *tag* in this document for clarity and in order to retain a consistent terminology with other documents in use by readers.

1.6.1 Vendor-defined Small Resource Items

These descriptors may be up to 8 bytes in length. Byte 0 has the value '01110xxx'b where bit 7=0 indicates that this is a small resource item, bits(6:3)=1110 is the small item name (0xE), and xxx is the number of bytes to follow (1-7). For IBM vendor defined tags, byte 1 is, by convention, a *type* byte which is used to distinguish between the tags. Type = 0x0 is reserved for non-IBM use.

1.6.1.1 Type=0x01: Chip Identification

This descriptor is used to identify a particular chip that implements a function when it is necessary to relate some chip-specific information. See the header file, **CHPIDPNP.H** (below) for details and chip identifiers.

```
#ifndef _CHPIDPNP_
#define _CHPIDPNP_

#define  ChipID_Packet     0x70         /* tag for ChipIDPack without size    */

/* The chipid is the vendor id followed by 4 hex digits, e.g., for IBM
   platforms: Chip_ID=IBMxxxx. To avoid confusion with PnP Device IDs, IBM
   chip ids will begin at 0x8000. Chip ids are assigned to ranges based on
   function (see below) and are unique to the chip, i.e. a chip may have
   multiple functions, but only one id */

typedef enum _Chip_ID {
```

```
/* Memory Controllers                    Memory Controller Range: IBM80xx   */
   Dakota = 0x8001,                   /* IBM8001:IBM North/South Dakota     */
   Idaho = 0x8002,                    /* IBM8002:IBM Idaho                  */
   Eagle = 0x8003,                    /* IBM8003:Motorola Eagle             */
   Kauai_Lanai = 0x8004,              /* IBM8004:IBM Kauai/Lanai            */
   Montana_Nevada = 0x8005,           /* IBM8005:IBM Montana/Nevada         */
   Union = 0x8006,                    /* IBM8006:IBM Union                  */
   Cobra_Viper = 0x8007,              /* IBM8007:IBM Cobra/Viper            */
   Grackle = 0x8008,                  /* IBM8008:Motorola Grackle           */

/* ISA Bridge Chips                      Bus Bridge Range: IBM81xx          */
   SIO_ZB = 0x8100,                   /* IBM8100:Intel 82378ZB              */
   FireCoral = 0x8101,                /* IBM8101:IBM FireCoral              */

/* PCI Bridge chips                      Bus Bridge Range: IBM81xx          */
/* Dakota = 0x8001,                      IBM8001:IBM North/South Dakota     */
/* Idaho = 0x8002,                       IBM8002:IBM Idaho                  */
/* Eagle = 0x8003,                       IBM8003:Motorola Eagle             */
/* Kauai_Lanai = 0x8004,                 IBM8004:IBM Kauai/Lanai            */
/* Montana_Nevada = 0x8005,              IBM8005:IBM Montana/Nevada         */
/* Union = 0x8006,                       IBM8006:IBM Union                  */
   Python = 0x8102,                   /* IBM8102:IBM Python                 */
   DEC21050 = 0x8103,                 /* IBM8103:PCI-PCI                    */
   IBM2782351 = 0x8106,               /* IBM8106:PCI-PCI                    */
   IBM2782352 = 0x8109,               /* IBM8109:PCI-PCI352                 */

/* PCMCIA Bridge Chips                   Bus Bridge Range: IBM81xx          */
   INTEL_8236SL = 0x8104,             /* IBM8104:Intel 8236SL               */
   RICOH_RF5C366C = 0x8105,           /* IBM8105:RICOH RF5C366C             */

/* EISA Bridge Chips                     Bus Bridge Range: IBM81xx          */
   INTEL_82374 = 0x8108,              /* IBM8108:Intel 82374/82375          */

/* MCA Bridge Chips                      Bus Bridge Range: IBM81xx          */
   MCACoral = 0x8107,                 /* IBM8107: 6xxMx - T=1 MCA (servers) */

/* L2 Cache controller                   L2 Controller Range: IBM82xx       */
   Cheyenne = 0x8200,                 /* IBM8200:IBM Cheyenne               */
   IDT = 0x8201,                      /* IBM8201:IDT                        */
   Sony1PB = 0x8202,                  /* IBM8202:Sony1PB                    */
   Mamba = 0x8203,                    /* IBM8203:IBM Mamba                  */
   Alaska = 0x8204,                   /* IBM8204:IBM Alaska                 */
   Glance = 0x8205,                   /* IBM8205:IBM Glance                 */
   Ocelot = 0x8206,                   /* IBM8206:IBM Ocelot                 */
/* Eagle = 0x8003,                       IBM8003:Motorola Eagle             */

/* Power management chips                PM Chip Range: IBM83xx             */
   Carrera = 0x8300,                  /* IBM8300:IBM Carrera                */
   Sig750 = 0x8301,                   /* IBM8301:Signetics 87C750           */

/* Interrupt Controller Chips            PIC Chip Range: IBM84xx            */
   MPIC_2 = 0x8400,                   /* IBM8400:IBM MPIC_2                 */

/* Miscellaneous Planar Chips            MISC Chip Range: IBM8Fxx           */
   DallasRTC = 0x8F00,                /* IBM8F00:Dallas 1385 compatible     */
   Dallas1585 = 0x8F01,               /* IBM8F01:Dallas 1585 compatible     */
   Timer8254 = 0x8F10,                /* IBM8F10: 8254-compatible timer     */
   HarddiskLt = 0x8FF0                /* IBM8FF0:Op Panel HD light          */

   } Chip_ID;

typedef struct _ChipIDPack {          /* This packet identifies chip set ID.*/
                                      /* Details of these specifics are     */
```

```
                                    /* documented in the chip specs.      */
   unsigned char  Tag;              /* small tag = 0x7n with n bytes      */
   unsigned char  Type;             /*    = 1 (Chip ID)                   */
   unsigned char  VendorID0;        /*    Bit(7)=0                        */
                                    /*    Bits(6:2)=1st character in      */
                                    /*       compressed ASCII             */
                                    /*    Bits(1:0)=2nd character in      */
                                    /*       compressed ASCII bits(4:3)   */
   unsigned char  VendorID1;        /*    Bits(7:5)=2nd character in      */
                                    /*       compressed ASCII bits(2:0)   */
                                    /*    Bits(4:0)=3rd character in      */
                                    /*       compressed ASCII             */
   unsigned char  Name[2];          /*    Chip ID name                    */
                                    /*                                    */
                                    /* Example: IBM8001                   */
                                    /*  I,B,M = 01001,00010,01101         */
                                    /*  bytes 0,1 = 00100100,01001101     */
                                    /*            =   2   4    4   D      */
                                    /*  byte 0 = 24                       */
                                    /*  byte 1 = 4D                       */
                                    /*  byte 2 = 01                       */
                                    /*  byte 3 = 80                       */
                                    /*                                    */

   } ChipIDPack;

#endif  /* ndef _CHPIDPNP_ */
```

1.6.1.2 Type=0x03: Processor Number

This descriptor defines the CPU number for a processor associated with a device array entry. See the header file, **CPNUMPNP.H** (below) for details.

```
/* 10/25/95                                                           */
/* Structure map for Processor Number in PnP Vendor specific packet   */

/* See Plug and Play ISA Specification, Version 1.0a, March 24, 1994. */
/* It (or later versions) is available on Compuserve in the PLUGPLAY  */
/* area.  This code has extensions to that specification, namely new  */
/* short and long tag types for platform dependent information        */
/* Warning: LE notation used throughout this file                     */

/* This descriptor is used to define the relationship between processors and  */
/* L2 caches.                                                                 */

#ifndef _CPNUMPNP_
#define _CPNUMPNP_

#define  CPNum_Packet      0x70         /* tag for CPNumPack without size     */

typedef struct _CPNumPack {
   unsigned char  Tag;                 /* small tag = 0x72                   */
   unsigned char  Type;                /*  = 3 (Processor Number Descriptor) */
   unsigned char  ProcessorNumber;
   } CPNumPack;

#endif  /* ndef _CHNUMPNP_ */
```

1.6.2 Vendor-defined Large Resource Items

These descriptors may be up to 64K bytes in length. Byte 0 has the value ‘10000100’b where bit 7=1 indicates that this is a large resource item, and bits(6:0)=0000100 is the large item name (0x4). The next two bytes indicate the length of the following data field, lower 8 bits followed by the upper 8 bits. For IBM vendor defined tags, byte 3 is, by convention, a *type* byte which is used to distinguish between the tags. Type = 0x0 is reserved for non-IBM use.

1.6.2.1 Type=0x01: Diskette Drives

This descriptor is used to describe the diskette drives attached to a diskette controller. It identifies the type of drive and whether it supports media sense, auto eject, and/or the alternate speed feature (for 4 MB 3.5” diskettes). See the header file **DSKTPNP.H** (below) for details .

```
#ifndef _DSKTPNP_
#define _DSKTPNP_
#define  Dskt_Packet 0x84               /* tag for DsktInfoPack            */
typedef enum _DISKETTE_FEATURE {
  MediaSense = 0x01,
  AutoEject = 0x02,
  AltSpeed = 0x04                       /* 10/20/94                        */
  } DISKETTE_FEATURE;
typedef struct _DsktDriveInfo {
   unsigned char  Dskt;                 /* diskette drives info array      */
                                        /*      0 : no drive present       */
                                        /*      1 : 3.5” 2MB drive         */
                                        /*      2 : 3.5” 4MB drive         */
                                        /*      3 : 5.25” 1.6MB drive      */
   unsigned char  Feature;              /* DISKETTE_FEATURE enum           */
   } DsktDriveInfo;

typedef struct _DsktInfoPack {
   unsigned char  Tag;                  /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;               /* lo byte of number of drives + 1  */
   unsigned char  Count1;               /* hi byte of number of drives + 1  */
   /* Count ((Count0 and Count1) - 1) / 2 = number of diskette drives       */
   unsigned char  Type;                 /*   = 1 (diskette)                */
   DsktDriveInfo  DDInfo[1];            /* drive infomation                */
   } DsktInfoPack;

#endif  /* ndef _DSKTPNP_ */
```

1.6.2.2 Type=0x02: L2 Cache

This descriptor gives parameters (size, algorithms, features) relating to the L2 cache included in the system. See the header file **L2PNP.H** (below) for details.

```
#ifndef _L2PNP_
#define _L2PNP_

#define  L2Info_Packet  0x84          /* tag for L2Info_Pack  */

typedef enum _L2_Store_Algorithm {
   None = 0,
   WriteThru = 1,                     /* Store Thru          */
```

```
   CopyBack = 2                         /* Store In             */
   } L2_Store_Algorithm;

typedef enum _L2_Cache_Asc {
   DirectMapped = 1,                     /* direct mapped */
   TwoWay = 2                            /* 2-way         */
   } L2_Cache_Asc;

typedef enum _L2_HW_Assist {
   Invalidate = 1,                      /* invalidate        */
   Flush = 2                            /* flush             */
   } L2_HW_Assist;

typedef struct _L2InfoPack {
   unsigned char  Tag;                   /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;                /* = 0x0D sizeof(L2InfoPack - 3)    */
   unsigned char  Count1;                /* = 0                 */
   unsigned char  Type;                  /* = 2 (L2 cache)      */
   unsigned char  L2_CacheSize[4];       /* In 1K bytes         */
   unsigned char  L2_CacheAsc[2];        /* L2_Cache_Asc enum   */
   unsigned char  L2_LineSize[2];
   unsigned char  L2_SectorSize[2];
   unsigned char  L2_StoreAlgorithm;     /* L2_Store_Algorithm enum */
   unsigned char  L2_HWAssist;           /* L2_HW_Assist            */
   } L2InfoPack;

#endif  /* ndef _L2PNP_ */
```

1.6.2.3 Type=0x03: PCI Bridge

This descriptor is used to pass parameters associated with a PCI bridge (identification, configuration address, and interrupt mapping). See the header file **PCIPNP.H** (below) for details.

```
#ifndef _PCIPNP_
#define _PCIPNP_

#define MAX_PCI_INTS    4

typedef enum _IntTypes {              /* interrupt cntlr types enumerator    */
   IntInvalid = 0,                    /*    invalid value                    */
   IntCtl8259 = 1,                    /*    8259                             */
   IntCtlMPIC = 2,                    /*    MPIC                             */
   IntCtlRS6K = 3                     /*    RS6K                             */
   } _IntTypes;

typedef struct _IntMap {              /* PCI Int to system conversion map    */
   unsigned char  SlotNumber;         /* First slot = 1; Integrated = Slot 0 */
   unsigned char  DevFuncNumber;      /* PCI slot's DeviceFunction number    */
   unsigned char  IntCtrlType;        /* Interrupt controller type           */
   unsigned char  IntCtrlNumber;      /* Interrupt controller number         */
                                      /*   0      8259 interrupt             */
                                      /*   0      MPIC interrupt             */
                                      /*   buid   RS6K interrupt             */
   unsigned short Int[MAX_PCI_INTS];  /* Interrupt mapping table             */
                                      /*    index 0 for INTA                 */
                                      /*    index 1 for INTB                 */
                                      /*    index 2 for INTC                 */
                                      /*    index 3 for INTD                 */
                                      /*    0xFFFF if not usable             */
                                      /*    0x8nnn if edge sensitive         */
```

```
   } IntMap;

typedef struct _PCIInfoPack {
   unsigned char  Tag;                 /* large tag = 0x84 Vendor specific    */
   unsigned char  Count0;              /* lo byte of count                    */
   unsigned char  Count1;              /* hi byte of count                    */
      /* count = number of PCI slots * sizeof(IntMap) + 21                    */
   unsigned char  Type;                /* = 3 (PCI bridge)                    */
   unsigned char  ConfigBaseAddress[8];/* Base address of PCI Configuration   */
                                       /*    system real address              */
   unsigned char  ConfigBaseData[8];   /* Base address of PCI Config data     */
                                       /*    system real address              */
   unsigned char  BusNumber;           /* PCI Bus number                      */
   unsigned char  Reserved[3];         /* Reserved (padded with 0)            */
   IntMap         Map[1];              /* Interrupt map array for each PCI    */
                                       /*  slots that are pluggable           */
                                       /*  number = (count-21)/sizeof(IntMap) */
   } PCIInfoPack;

#endif  /* ndef _PCIPNP_ */
```

1.6.2.4 Type=0x04: Display

This descriptor is used to identify the type of display and various characteristics (resolution, bpp) of displays attached to portable systems. See the header file **DISPPNP.H** (below) for details.

```
#ifndef _DISPPNP_
#define _DISPPNP_

#define  Disp_Packet 0x84             /* tag for DisplayInfoPack            */

typedef enum _Display_Mode {
   Console_Text      = 0,             /* Video in VGA text mode             */
   Console_Video     = 1,             /* Video in 8 bpp graphics            */
   Console_Video16   = 2,             /* Video in 5,6,5 graphics            */
   Console_Video24   = 3,             /* Video in 8,8,8 graphics            */
   Console_Video32   = 4
   } Display_Mode;

typedef enum _Display_ID {
   UnknownDisplay    = 0x00,
   IBM_F8560         = 0xfa,          /* IBM F8560 12.1" Color TFT 1024x768 */
   IBM_F8550         = 0xfb,          /* IBM F8550 12.1" Color TFT  800x600 */
   IBM_F8532         = 0xfc,          /* IBM F8532 10.4" Color TFT  800x600 */
   TOSHIBA_STNC      = 0xfd,          /* Toshiba   10.4" Color DSTN 640x480 */
   IBM_F8515         = 0xfe,          /* IBM F8515 10.4" Color TFT  640x480 */
   NoDisplay         = 0xff
   } Display_ID;



typedef struct _DispInfoPack {
   unsigned char  Tag;                 /* large tag = 0x84 Vendor specific   */
   unsigned char  Count0;              /* = 9                                */
   unsigned char  Count1;              /* = 0                                */
   unsigned char  Type;                /* = 4 (display)                      */
   unsigned char  DisplayMode;         /* Display_Mode enum                  */
   unsigned char  DisplayID;           /* Display_ID enum                    */
   unsigned char  Hor_Pixels[2];       /* Horizontal resolution, e.g. 640    */
   unsigned char  Ver_Pixels[2];       /* Vertical Resolution, e.g. 480      */
   unsigned char  BytesPerRow[2];      /* Number of bytes per VRAM row       */
```

```
   unsigned char  VRAMAddr[8];         /* Location of video buffer          */
   unsigned char  VRAMSize[8];         /* Size of video buffer              */
   } DispInfoPack;

#endif  /* ndef _DISPPNP_ */
```

1.6.2.5 Type=0x05: Bridge Address Translation

This descriptor is used to define address translations performed by a bus bridge. Bus bridges will generally observe addresses that appear on the parent and child busses and make a determination as to whether to respond to the address and, if so, whether to apply a translation to the address. The combined effect of these translations determine the physical memory map of the system.

This descriptor provides for the specification of various types of decoding and address mapping that might be implemented in a bus bridge. See the header file **BRATPNP.H** (below) for details.

- Decoding
  - — Positive - particular addresses or ranges of addresses are decoded by the bridge and passed on either directly or translated.
  - — Subtractive - particular addresses or ranges of addresses are decoded by the bridge only after a determination is made that the address cycle was not accepted by another agent on the bus.
- Translation Type
  - — Direct - the new address is generated by applying a simple offset to the original address.
  - — Mapped - a translation table is used, where each address can potentially become a unique new address after translation.
  - — PowerPC Special Storage Segment Translation

Note: Need to write about the “private bit”.

- Address Conversion
  - — PIO_BM - system addressing of bus memory.
  - — PIO_BIO - system addressing of bus I/O.
  - — DMA_BM_Parent - DMA from bus memory to parent bus.
- Addresses
  - — Parent Base Address - for a mapped translation type, this is the parent bus address of the translation table.
  - — Bus Base Address - sibling bus
  - — Bus Address Range - sibling bus address range.

```
#ifndef _BRATPNP_
```

```
#define _BRATPNP_

#define  BRAT_Packet  0x84              /* tag for BRAT_Pack               */

typedef enum _Translation_Flags {
   positive_decode = 0x01               /* 0x0 = subtractive decode        */
                                        /* bits[7:1] are reserved          */
   } Translation_Flags;

typedef enum _Translation_Type {
   direct = 1,                          /* direct translation              */
   mapped = 2,                          /* mapped translation              */
   PPC_Spec_Seg = 3                     /* PowerPC special storage segment
                                           (T=1) translation               */
   } Translation_Type;

typedef enum _Address_Conversion {
   PIO_BM = 1,                          /* PIO system to bus memory        */
   PIO_BIO = 2,                         /* PIO system to bus I/O           */
   DMA_BM_Parent = 3                    /* DMA bus memory to parent bus    */
   } Address_Conversion;

typedef struct _BRATPack {
   unsigned char  Tag;                  /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;               /* = 0x1C sizeof(BRATPack)-3       */
   unsigned char  Count1;               /* = 0                             */
   unsigned char  Type;                 /* = 5                             */
   unsigned char  BRAT_Flags;           /* Translation_Flags               */
   unsigned char  BRAT_Type;            /* Translation_Type                */
   unsigned char  BRAT_Addr_Conv;       /* Address Conversion types        */
   unsigned char  Reserved;
   unsigned char  ParentBaseAddress[8];
   unsigned char  BusBaseAddress[8];
   unsigned char  BusAddressRange[8];
   } BRATPack;

#endif  /* ndef _BRATPNP_ */
```

1.6.2.6 Type=0x06: Bus Bridge Attributes

This descriptor gives the sibling bus speed and number of slots. See the header file **BBATRPNP.H** (below) for details.

```
#ifndef _BBATRPNP_
#define _BBATRPNP_

#define  BBATR_Packet  0x84             /* tag for BBATR_Pack  */

typedef struct _BBATRPack {
   unsigned char  Tag;                  /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;               /* = 0x06 sizeof(BBATRPack - 3)     */
   unsigned char  Count1;               /* = 0                 */
   unsigned char  Type;                 /* = 6                 */
   unsigned char  BusSpeed[4];          /* bus speed in Hz                  */
   unsigned char  NumSlots;             /* number of bus slots              */
                                        /* ISA: FF = some number of slots   */
                                        /*           are present.           */
                                        /*      00 = there are no slots     */
   } BBATRPack;
```

```
#endif  /* ndef _BBATRPNP_ */
```

1.6.2.7 Type=0x07: SCSI Controller Information

This descriptor gives the SCSI identifier for each SCSI bus connected to a controller. See the header file **SCSIPNP.H** (below) for details.

```
#ifndef _SCSIPNP_
#define _SCSIPNP_

#define  SCSICtlInfo_Packet  0x84       /* tag for SCSIControllerPack       */

typedef struct _SCSIControllerPack {
   unsigned char  Tag;                  /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;               /* = sizeof(SCSIControllerPack) - 3 */
   unsigned char  Count1;               /*   + BusCount - 1                 */
   unsigned char  Type;                 /* = 7 (SCSI Controller)            */
   unsigned char  BusCount;             /* Number of SCSI busses on device  */
   unsigned char  SCSIID[1];            /* Array with size of BusCount.     */
                                        /* Each entry contains SCSI ID of   */
                                        /* the indexed SCSI bus             */
   } SCSIControllerPack;

#endif  /* ndef _SCSIPNP_ */
```

1.6.2.8 Type=0x08: Power Management Support

This descriptor gives the supported attributes of the system's power management implementation. See the attributes table and the header file **PMSPTPNP.H** (below) for details.

Table 7. Power Management Attributes

PMattribute	Byte/bit	Name	Description
0x0001	B4/b0	hibernation_support	Firmware supports resume from hibernation (all system components other than specific PM hardware is powered off).
0x0002	B4/b1	suspend_support	Hardware supports a mode in which all but main memory is powered off. Firmware supports resume from this state.
0x0004	B4/b2	LID_support	Interrupts are presented when a lid (portable systems) is opened or closed so that a PM system can take appropriate action.
0x0008	B4/b3	battery_support	The system is capable of running off a battery and there are battery power management capabilities implemented (system dependent).
0x0010	B4/b4	ringing_resume_from_hibernation	Modem ring can trigger resume from hibernation.
0x0020	B4/b5	ringing _resume_from_suspend	Modem ring can trigger resume from suspend.

Table 7. Power Management Attributes

PMattribute	Byte/bit	Name	Description
0x0040	B4/b6	resume_from_hibernation_at_specified_time	An alarm can be set to trigger resume from hibernation.
0x0080	B4/b7	resume_from_suspend_at_specified_time	An alarm can be set to trigger resume from suspend.
0x0100	B5/b0	hibernation_from_suspend_at_specified_time	An alarm can be set to trigger a change from suspended state to a hibernated state..
0x0200	B5/b1	software_controllable_main_power_switch	The main power switch does not directly turn on/off power to the system, but is visible to the software.
0x0400	B5/b2	general_purpose_input_pin_for_resume_trigger	Some other input can trigger resume.
0x0800	B5/b3	fail_safe_hardware_main_power_off_switch	If a software controllable main power switch is implemented, there is a hardware mechanism to manually turn on/off power should the software mechanism fail.
0x1000	B5/b4	resume_button_support	There is a button that can trigger the hardware to resume.
0x2000	B5/b5	inhibit_auto_transition	Automatic transistion between PM states is inhibited.

```
#ifndef _PMSPTPNP_
#define _PMSPTPNP_

#define  PMsupport_Packet  0x84         /* tag for PMsupportPack            */

typedef enum _PM_attribute {
   hibernation_support = 0x0001,
   suspend_suppport = 0x0002,
   LID_support = 0x0004,
   battery_support = 0x0008,
   ringing_resume_from_hibernation = 0x0010,
   ringing_resume_from_suspend = 0x0020,
   resume_from_hibernation_at_specified_time = 0x0040,
   resume_from_suspend_at_specified_time = 0x0080,
   hibernation_from_suspend_at_specified_time = 0x0100,
   software_controllable_main_power_switch = 0x0200,
   general_purpose_input_pin_for_resume_trigger = 0x0400,
   fail_safe_hardware_main_power_off_switch = 0x0800,
   resume_button_support = 0x1000,
   inhibit_auto_transition = 0x2000
   } PM_attribute;

typedef struct _PMsupportPack {
   unsigned char  Tag;                  /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;               /* = 0x05 sizeof(PMsupprtPack)-3    */
   unsigned char  Count1;               /* = 0                              */
   unsigned char  Type;                 /* = 8 (PM Controller)              */
   unsigned long  PMattribute;
   } PMsupportPack;

#endif  /* ndef _PMSPTPNP_ */
```

1.6.2.9 Type=0x09: Generic Address

This descriptor permits specification of addresses beyond what is provided by the standard PnP address tags. See the header file **GENADRPNP.H** (below) for details.

- Byte 5: AddrInfo is the number of bits decoded by the hardware for this range of addresses.
- Byte 8: Address[0] is base address bits(7:0), etc., little-endian byte ordering within the doubleword represented by Address[8] (bytes 8-15). The same holds true for Length[8].

```
#ifndef _GENADPNP_
#define _GENADPNP_

typedef enum _AddrType {
   ISAIOAddr = 1,                    /* ISA/PCI I/O address                */
   IOMemAddr = 2,                    /* I/O Memory address space           */
   SysMemAddr = 3                    /* System memory address              */
} AddrType;

typedef struct _GenericAddr {
   unsigned char  Tag;               /* large tag = 0x84 Vendor specific   */
   unsigned char  Count0;            /* lo byte of count                   */
   unsigned char  Count1;            /* hi byte of count                   */
                                     /*   count = sizeof(_GenericAddr) - 3 */
   unsigned char  Type;              /* = 9 (Generic Address)              */
   unsigned char  AddrType;          /* see enum AddrType                  */
   unsigned char  AddrInfo;          /* info for the AddrType              */
                                     /*    I/O address type: number of bits */
   unsigned char  Reserved[2];       /* reserved                           */
   unsigned char  Address[8];        /* Address                            */
   unsigned char  Length[8];         /* Number of contiguous bytes used    */
} GenericAddr;

#endif  /* ndef _GENADPNP_ */
```

1.6.2.10 Type=0x0A: ISA Bridge Information

This tag describes how ISA interrupts are mapped to system interrupts. See the header file **ISAPNP.H** (below) for details.

- Byte 5: IntCtrlNumber is provided to support the RS/6000 interrupt structure which needs to specify a BUID.
- Bytes 6&7: Int[0] is a two byte representation of the system interrupt to which IRQ0 is mapped - in little-endian byte ordering within the halfword (bytes 6&7). Byte 6 is “system interrupt number” bits(7:0); byte 7 is “system interrupt number” bits(15:8).The same holds true for Int[1 thru MAX_ISA_INTS].

```
#ifndef _ISAPNP_
#define _ISAPNP_

#define MAX_ISA_INTS    16
```

```
typedef struct _ISAInfoPack {
   unsigned char  Tag;                 /* large tag = 0x84 Vendor specific   */
   unsigned char  Count0;              /* lo byte of count                   */
   unsigned char  Count1;              /* hi byte of count                   */
                                       /*   count = sizeof(_ISAInfoPack) - 3 */
   unsigned char  Type;                /* = 0x0A (ISA bridge)                */
   unsigned char  IntCtrlType;         /* Interrupt controller type          */
                                       /* enum _IntTypes in pcipnp.h         */
   unsigned char  IntCtrlNumber;       /* Interrupt controller number        */
                                       /*   0      8259 interrupt            */
                                       /*   0      MPIC interrupt            */
                                       /*   buid   RS6K interrupt            */
   unsigned short Int[MAX_ISA_INTS];   /* Interrupt mapping table            */
                                       /*   index  0 for IRQ0                */
                                       /*   index  1 for IRQ1                */
                                       /*   index 15 for IRQ15               */
                                       /*   0xFFFF if not usable             */
} ISAInfoPack;

#endif  /* ndef _ISAPNP_ */
```

1.6.2.11 Type=0x0B: Video Channels

This descriptor gives characteristics of video capture support. See the header file **VIDCHPNP.H** (below) for details.

```
/* If this packet is not available on a system with a G10 video capture       */
/* card installed, assume WOODFIELD (ThinkPad Power Series 850 - M/T 6020     */
/* developer's kit).  If this packet is not available on a system without a   */
/* G10 video capture card, assume no video channels are available except for  */
/* VGA graphics.                                                              */

#ifndef _VIDCHPNP_
#define _VIDCHPNP_

/* Physical Video Channel ID definitions                                      */
/* Note that a physical channel may have more than one logical channel.  For  */
/* example, a video input channel can be used for NTSC and PAL.  In another   */
/* example, a video channel can be used for composite video and S-video.      */
/*                                                                            */
/* At this moment, only Brooktree Bt812 and ASCII V7310A are the available    */
/* video chips.  Originator's intention is that the upper 4 bits represent    */
/* controller types and the lower 4 bits represent controller-dependent video */
/* channel names.  But, for flexibility, the above guideline is not           */
/* registered at this revision.  It may be applied in the future.  Note that  */
/* Extension = 0xFF is reserved for future uses.                              */

typedef enum _PhysicalChannelID {
   ViD0_Y              = 0x00,         /* Bt812 ViD0_Y                       */
   ViD0_C              = 0x01,         /* Bt812 ViD0_C                       */
   ViD1_Y              = 0x02,         /* Bt812 ViD1_Y                       */
   ViD1_C              = 0x03,         /* Bt812 ViD1_C                       */
   ViD2_Y              = 0x04,         /* Bt812 ViD2_Y                       */
   ViD2_C              = 0x05,         /* Bt812 ViD2_C                       */
   ViD3_Y              = 0x06,         /* Bt812 ViD3_Y                       */
   ViD3_C              = 0x07,         /* Bt812 ViD3_C                       */
   VoR                 = 0x10,         /* V7310A  VoR                        */
   VoG                 = 0x11,         /* V7310A  VoG                        */
   VoB                 = 0x12,         /* V7310A  VoB                        */
   Extension           = 0xFF,         /* Reserved for future use            */
```

```
   } PhysicalChannelID;

typedef enum _ConnectorType {
   RCAJack             = 0x00,         /* RCA Jack (1-pin)                  */
   MiniDIN             = 0x01,         /* Mini DIN Connector (4-pin)        */
   BNC                 = 0x02,         /* BNC Coaxial Connector             */
   VGADSub             = 0x03,         /* VGA D-Sub Connector (15-pin)      */
   T6040Camera         = 0x04,         /* WOODFIELD Camera Connector        */
   InternalConnection  = 0xFE,         /* No external connectors            */
   NotConnected        = 0xFF,         /* No connection at all              */
   } ConnectorType;

typedef enum _SignalType {
   NTSCComposite    = 0x00,            /* NTSC Composite Video              */
   NTSCLume         = 0x01,            /* NTSC S-Video Luma                 */
   NTSCChroma       = 0x02,            /* NTSC S-Video Chroma               */
   NTSCY            = 0x03,            /* NTSC Component Y                  */
   NTSCYR           = 0x04,            /* NTSC Component Y-R                */
   NTSCYB           = 0x05,            /* NTSC Component Y-B                */
   NTSCTimingOnly   = 0x06,            /* NTSC Timing Only                  */
   NTSCDiagnostics  = 0x07,            /* NTSC Wrap-back                    */
   PALComposite     = 0x10,            /* PAL Composite Video               */
   PALLume          = 0x11,            /* PAL S-Video Luma                  */
   PALChroma        = 0x12,            /* PAL S-Video Chroma                */
   PALY             = 0x13,            /* PAL Component Y                   */
   PALYR            = 0x14,            /* PAL Component Y-R                 */
   PALYB            = 0x15,            /* PAL Component Y-B                 */
   PALTimingOnly    = 0x16,            /* PAL Timing Only                   */
   PALDiagnostics   = 0x17,            /* PAL Wrap-back                     */
   SECAMComposite   = 0x20,            /* SECAM Composite Video             */
   SECAMLume        = 0x21,            /* SECAM S-Video Luma                */
   SECAMChroma      = 0x22,            /* SECAM S-Video Chroma              */
   SECAMY           = 0x23,            /* SECAM Component Y                 */
   SECAMYR          = 0x24,            /* SECAM Component Y-R               */
   SECAMYB          = 0x25,            /* SECAM Component Y-B               */
   SECAMTimingOnly  = 0x26,            /* SECAM Timing Only                 */
   SECAMDiagnostics = 0x27,            /* SECAM Wrap-back                   */
   VGARed           = 0x30,            /* VGA Red                           */
   VGAGreen         = 0x31,            /* VGA Green                         */
   VGABlue          = 0x32,            /* VGA Blue                          */
   NoSignal         = 0xFF,            /* No Signal                         */
   } SignalType;

typedef struct _VidChanMap {           /* Video Channel Map                 */
   unsigned char  LogocialChannelID;   /* Video Channel = 0, 1, ... n       */
   unsigned char  VideoIO;             /* 0 : In, 1 : Out                   */
                                       /* Note that a single physical channel */
                                       /* may support both video-in and     */
                                       /* video-out future implementations.  */
                                       /* Under such circumstances, separate */
                                       /* logical channel IDs must be       */
                                       /* assigned.                         */
   unsigned char  PhysicalChannelID;   /* Controller dependent port ID      */
   unsigned char  ConnectorType;       /* Mechanical Connector Type         */
   unsigned char  SignalType;          /* Electrical Signal Type            */
   } VidChanMap;

typedef struct _VidChanInfoPack {
   unsigned char  Tag;                 /* large tag = 0x84 Vendor specific  */
   unsigned char  Count0;              /* lo byte of count                  */
   unsigned char  Count1;              /* hi byte of count                  */
  /* count = number of logical video channels * sizeof(VidChanMap) + 1      */
   unsigned char  Type;                /* = B                               */
   VidChanMap     VCMap[1];            /* Video channel map                 */
```

```
                                        /*                                  */
   } VidChanInfoPack;

#endif  /* ndef _VIDCHPNP_ */
```

1.6.2.12 Type=0x0C: Power Control

This descriptor lists power control capabilities. See the header file **PCSPTPNP.H** (below) for details.

```
#ifndef _PCSPTPNP_
#define _PCSPTPNP_

#define  PCsupport_Packet  0x84         /* tag for PCsupportPack     */

typedef enum _PC_attribute {
   program_power_off_support = 0x0001,
   program_power_on_support  = 0x0002
   } PC_attribute;

typedef struct _PCsupportPack {
   unsigned char  Tag;                  /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;               /* = 0x05 sizeof(PCsupprtPack)-3    */
   unsigned char  Count1;               /* = 0                              */
   unsigned char  Type;                 /* = 0x0C (PC Controller)           */
   unsigned long  PCattribute;
   } PCsupportPack;

#endif  /* ndef _PCSPTPNP_ */
```

1.6.2.13 Type=0x0D: Memory SIMM PD Data

This descriptor gives SIMM presence detect data. There is one byte of PD data per SIMM connected to the memory controller listing this descriptor; the meaning of the bits within the byte is system specific. See the header file **MEMPDPNP.H** (below) for details.

```
#ifndef _MEMPDPNP
 #define _MEMPDPNP

 #define  MemPD_Packet  0x84

 typedef struct _MemPDPack {
    unsigned char  Tag;                 /* large tag = 0x84 Vendor specific   */
    unsigned char  Count0;              /* lo byte of count                   */
    unsigned char  Count1;              /* hi byte of count                   */
    unsigned char  Type;                /* = 0xD (Memory SIMM PD Descriptor)  */
    unsigned char  PDBits[1];           /* array for SIMM PD bytes            */
    // SIMM PD info notes:
    // 1. The actual runtime length, n, of the PDBits array is
    //    n = RESIDUAL.VitalProductData.NumSIMMSlots * _Banks
    // 2. For Victory products, _Banks = 1
    // 3. A value of 0xFF for PDBits[m] indicates SIMM slot m does not contain
    //    a SIMM (m refers to logical, zero based SIMM slot numbers. it does
    //    not imply that the number on or next to the SIMM connector has the
    //    same value. this file does not define the physical SIMM numbering)
    } MemPDPack;
```

```
#endif
```

1.6.2.14 Type=0x0E: System Interrupts

This descriptor can be attached to a device to specifiy an interrupt *not* connected to an AT-style interrupt tree (use PnP small resource item *IRQ format* for AT interrupts). See the header file **SINTPNP.H** (below) for details. IntCtrlType and IntCtrlNumber contain values as in ISAPNP.H.

```
#ifndef _SINTPNP_
#define _SINTPNP_

#define  SINT_Packet  0x84

typedef struct _SysIntPack {
   unsigned char  Tag;                 /* large tag = 0x84 Vendor specific   */
   unsigned char  Count0;              /* lo byte of count                   */
   unsigned char  Count1;              /* hi byte of count                   */
                                       /*   count = sizeof(SysIntPack) - 3   */
   unsigned char  Type;                /* = 0xE (System Interrupt Descriptor)*/
   unsigned char  IntCtrlType;         /* Interrupt controller type          */
   unsigned char  IntCtrlNumber;       /* Interrupt controller number        */
   unsigned char  Int[2];              /* Interrupt number                   */
                                       /* Int[0] = bits[7:0]                 */
                                       /* Int[1] = bits[15:8]                */
                                       /* high order bit (bit 15) info:      */
                                       /* 1 => edge sensitive; 0 => level    */
   } SysIntPack;

#endif
```

1.6.2.15 Type=0x0F: Error Log

This descriptor provides for an error log embedded in the DevicePnPHeap. See the header file **ERRLGPNP.H** (below) for details.

```
#ifndef _ERRLGPNP
#define _ERRLGPNP

#define RAMErrorLog_Packet  0x84

 typedef struct _RAMErrorLogPack {
    unsigned char  Tag;                 /* large tag = 0x84 Vendor specific   */
    unsigned char  Count0;              /* lo byte of count                   */
    unsigned char  Count1;              /* hi byte of count                   */
    unsigned char  Type;                /* = 0xF (RAM Error Log Descriptor)   */
    unsigned char  ErrLog[1];           /* array for Error Log bytes          */
    // Error Log notes:
    // 1. This error log is present if and only if
    //    RESIDUAL.VitalProductData.RAMErrorLogOffset != NULL
    // 2. If present, this log is on the PNP heap at location
    //    RESIDUAL.DevicePnPHeap[RESIDUAL.VitalProductData.RAMErrorLogOffset]
    // 3. This error log is a duplicate of the nvram error log with the
    //    following difference: it is cleared at the beginning of each system
    //    boot. Hence, unlike the nvram error log, the first error in this log
    //     will be the first error for the  current boot. The second entry will
```

```
    //     be the last error, in the same manner as the nvram error log
    } RAMErrorLogPack;

#endif
```

1.6.2.16 Type=0x10: Extended VPD

This descriptor provides for extended vital product data embedded in the DevicePnPHeap. See the header file **EXVPDPNP.H** (below) for details.

```
#ifndef _EXVPDPNP
#define _EXVPDPNP

#define ExtVPD_Packet  0x84

typedef struct _ExtVPDPack {
   unsigned char  Tag;                /* large tag = 0x84 Vendor specific   */
   unsigned char  Count0;             /* lo byte of count                   */
   unsigned char  Count1;             /* hi byte of count                   */
   unsigned char  Type;               /* = 0x10 (Extended VPD Descriptor)   */
   unsigned char  vpd_start;          /* undefined                          */
   unsigned char  Flag[3];            /* contains "VPD" in ascii format     */
                                      /* see the notes below for detailed   */
                                      /* info on the following fields       */
   unsigned char  LenHI;              /* length (high byte) of data below   */
   unsigned char  LenLO;              /* length (low  byte) of data below   */
   unsigned char  NextOffsetHI;       /* offset to the next VPD block, a 0  */
   unsigned char  NextOffsetLO;       /* value indicates no more blocks     */
   unsigned char  VPDBlockData[1];    /* array for Extended VPD             */

   // Extended VPD notes:
   // 1. This PNP structure contains an array of VPD data, herein referred to
   //    as "extended VPD". Extended in the sense that a service processor is
   //    required to extract the VPD from various "extended locations" within
   //    the system hardware (e.g., from the io planar, the processor card,
   //    the service processor itself, etc). If more than one extended VPD
   //    block is present, each one will be encapsulated in one of these PNP
   //    structures, with each structure catenated to the previous structure
   //    as explained inherently from the notes below. The extended VPD block
   //    begins at the "vpd_start" field of each present PNP structure.
   // 2. The first location of each VPD block will begin with its own
   //    "vpd_start" field (followed by its own Flag[], LenHI, .. etc). The
   //    "NextOffsetHI/LO" value indicates if additional VPD Blocks are
   //    present. A value of 0 defines the current block as the last. If the
   //    value is non-zero, the beginning "vpd_start" field of the next block
   //    will be "NextOffsetHI/LO" bytes from the current "vpd_start" field
   //    location.
   // 3. The "LenHI/LO" field is the length of the data for the current
   //    block, beginning with the VPDBlockData[0] byte of the current Block.
   //    WARNING: The unit value of this field represents two bytes. Hence,
   //    the total length of the current block data array field, in bytes,
   //    is two times "LenHI/LO".
   // 4. Extended VPD data will be present if and only if
   //    RESIDUAL.VitalProductData.ExtendedVPD != NULL in which case
   //    RESIDUAL.DevicePnPHeap[RESIDUAL.VitalProductData.ExtendedVPD]
   //    is the "vpd_start" field of the first VPD data block.
   //
   // 5. The data array (ExtVPDBlockData[]) contains data in the format of
   //    the RISC/6000 system VPD. E.G., "*XYLenSpecific-data". The asterisk
   //    is always present at the beginning of each field. XY represents an
   //    ASCII alphanumeric sequence of two characters that describe the
   //    "Specific-data" field. Len is a one byte unsigned field representing
   //    the total length of the data field, beginning with the * and ending
   //    with the last byte of the Specific-data, divided by two. In other
   //    words, the total length of the field is "Len" two-byte words. 2 is
```

```
   //    the minimum value for Len (in which case there is no Specific-data).
   //    Except for the "Len" field byte, which is in binary, all other bytes
   //    are in ASCII.
   //
   //    Asside note:
   //    The complete compliment of VPD can be obtained without any knowledge
   //    of PNP format. Notes 4 & 2 tell how to get to the first VPD block
   //    and how to find any additional VPD blocks, note 5 gives the format
   //    of the VPD in each VPD block, and note 3 gives the length of each
   //    VPD block. The rest of the fields, e.g. Tag, Count0, and Count1,
   //    conform to PNP specifications not defined in this header file, and
   //    may be ignored if all one needs is to extact the extended VPD.
   } ExtVPDPack;

#endif
```

1.6.2.17 Type=0x11: Timebase Control

This descriptor specifies the register address and bit that is used to enable/disable the timebase. See the header file **TBCTLPNP.H** (below) for details.

```
/* 10/10/95                                                              */
/* Structure map for Timebase Control                                    */

/* See Plug and Play ISA Specification, Version 1.0a, March 24, 1994.    */
/* It (or later versions) is available on Compuserve in the PLUGPLAY     */
/* area.  This code has extensions to that specification, namely new     */
/* short and long tag types for platform dependent information           */

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

#ifndef _TBCTLPNP_
#define _TBCTLPNP_

#define  TBCtl_Packet   0x84

/* This packet specifies the register address and bit that is used to
   enable/disable the timebase. */

typedef struct _TBCtlPack {
   unsigned char  Tag;                 /* large tag = 0x84 Vendor Specific   */
   unsigned char  Count0;              /* lo byte of count = 0xD             */
   unsigned char  Count1;              /* hi byte of count = 0x0             */
   unsigned char  Type;                /* = 0x11 (Timebase Control)          */
   unsigned char  TBCtlBit;            /* The bit number (in HEX) in the     */
                                       /* register at TBCtlAddr that         */
                                       /* enables/disables the Timebase.     */
                                       /* Bit 0 is low order (2**0).         */
   unsigned char  TBEnable;            /* The boolean state of TBCtlBit that */
                                       /* means "enable" (0x0 or 0x1)        */
   unsigned char  Reserved[2];         /* set to 0x0                         */
   unsigned char  TBCtlAddr[8];        /* The address of the register        */
                                       /* containing the Timebase            */
                                       /* enable/disable bit.                */
   } TBCtlPack;

#endif
```

residual.h Appendix A

```
/* 5/04/95                                                                  */
/*--------------------------------------------------------------------------*/
/*      Residual Data header definitions and prototypes                     */
/*--------------------------------------------------------------------------*/

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

#ifndef _RESIDUAL_
#define _RESIDUAL_

#define MAX_CPUS 32                    /* These should be set to the maximum */
#define MAX_MEMS 64                    /* number possible for this system.   */
#define MAX_DEVICES 256                /* Changing these will change the     */
#define AVE_PNP_SIZE 32                /* structure, hence the version of    */
#define MAX_MEM_SEGS 64                /* this header file.                  */

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

#include "pnp.h"

typedef enum _L1CACHE_TYPE {
  NoneCAC = 0,
  SplitCAC = 1,
  CombinedCAC = 2
  } L1CACHE_TYPE;

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

typedef enum _FIRMWARE_SUPPORT {
  Conventional = 0x01,
  OpenFirmware = 0x02,
  Diagnostics = 0x04,
  LowDebug = 0x08,
  Multiboot = 0x10,
  LowClient = 0x20,
  Hex41 = 0x40,
  FAT = 0x80,
```

```
  ISO9660 = 0x0100,
  SCSI_InitiatorID_Override = 0x0200,
  Tape_Boot = 0x0400,
  FW_Boot_Path = 0x0800
  } FIRMWARE_SUPPORT;

typedef enum _FIRMWARE_SUPPLIERS {
  IBMFirmware = 0x00,
  MotoFirmware = 0x01,
  FirmWorks = 0x02,
  Bull = 0x03,
  } FIRMWARE_SUPPLIERS;

typedef enum _ENDIAN_SWITCH_METHODS {
  UsePort92 = 0x01,
  UsePCIConfigA8 = 0x02,
  UseFF001030 = 0x03,
  } ENDIAN_SWITCH_METHODS;

typedef enum _SPREAD_IO_METHODS {
  UsePort850 = 0x00,
/*UsePCIConfigA8 = 0x02,*/
  } SPREAD_IO_METHODS;

typedef struct _VPD {

  /* Box dependent stuff */
  unsigned char PrintableModel[32];     /* Null terminated string.
                                           Must be of the form:
                                           vvv,<20h>,<model designation>,<0x0>
                                           where vvv is the vendor ID
                                           e.g. IBM PPS MODEL 6015<0x0>       */
  unsigned char Serial[16];             /* 12/94:
                                           Serial Number; must be of the form:
                                           vvv<serial number> where vvv is the
                                           vendor ID.
                                           e.g. IBM60151234567<20h><20h>      */
  unsigned char Reserved[48];
  unsigned long FirmwareSupplier;       /* See FirmwareSuppliers enum         */
  unsigned long FirmwareSupports;       /* See FirmwareSupport enum           */
  unsigned long NvramSize;              /* Size of nvram in bytes             */
  unsigned long NumSIMMSlots;
  unsigned short EndianSwitchMethod;    /* See EndianSwitchMethods enum       */
  unsigned short SpreadIOMethod;        /* See SpreadIOMethods enum           */
  unsigned long SmpIar;
  unsigned long RAMErrorLogOffset;      /* Heap offset to error log           */
  unsigned long Reserved5;
  unsigned long Reserved6;
  unsigned long ProcessorHz;            /* Processor clock frequency in Hertz */
  unsigned long ProcessorBusHz;         /* Processor bus clock frequency      */
  unsigned long Reserved7;
  unsigned long TimeBaseDivisor;        /* (Bus clocks per timebase tic)*1000 */
  unsigned long WordWidth;              /* Word width in bits                 */
  unsigned long PageSize;               /* Page size in bytes                 */
  unsigned long CoherenceBlockSize;     /* Unit of transfer in/out of cache
                                           for which coherency is maintained;
                                           normally <= CacheLineSize.         */
  unsigned long GranuleSize;            /* Unit of lock allocation to avoid   */
                                        /*   false sharing of locks.          */

  /* L1 Cache variables */
  unsigned long CacheSize;              /* L1 Cache size in KB. This is the   */
                                        /*   total size of the L1, whether    */
                                        /*   combined or split                */
```

```
  unsigned long CacheAttrib;            /* L1CACHE_TYPE                       */
  unsigned long CacheAssoc;             /* L1 Cache associativity. Use this
                                           for combined cache. If split, put
                                           zeros here.                        */
  unsigned long CacheLineSize;          /* L1 Cache line size in bytes. Use
                                           for combined cache. If split, put
                                           zeros here.                        */
  /* For split L1 Cache: (= combined if combined cache) */
  unsigned long I_CacheSize;
  unsigned long I_CacheAssoc;
  unsigned long I_CacheLineSize;
  unsigned long D_CacheSize;
  unsigned long D_CacheAssoc;
  unsigned long D_CacheLineSize;

  /* Translation Lookaside Buffer variables */
  unsigned long TLBSize;                /* Total number of TLBs on the system */
  unsigned long TLBAttrib;              /* Combined I+D or split TLB          */
  unsigned long TLBAssoc;               /* TLB Associativity. Use this for
                                           combined TLB. If split, put zeros
                                           here.                              */
  /* For split TLB: (= combined if combined TLB) */
  unsigned long I_TLBSize;
  unsigned long I_TLBAssoc;
  unsigned long D_TLBSize;
  unsigned long D_TLBAssoc;

  unsigned long ExtendedVPD;            /* Offset to extended VPD area;
                                           null if unused                     */
  } VPD;

typedef enum _DEVICE_FLAGS {
  Enabled = 0x4000,                     /* 1 - PCI device is enabled          */
  Integrated = 0x2000,
  Failed = 0x1000,                      /* 1 - device failed POST code tests  */
  Static = 0x0800,                      /* 0 - dynamically configurable
                                           1 - static                         */
  Dock = 0x0400,                        /* 0 - not a docking station device
                                           1 - is a docking station device    */
  Boot = 0x0200,                        /* 0 - device cannot be used for BOOT
                                           1 - can be a BOOT device           */
  Configurable = 0x0100,                /* 1 - device is configurable         */
  Disableable = 0x80,                   /* 1 - device can be disabled         */
  PowerManaged = 0x40,                  /* 0 - not managed; 1 - managed       */
  ReadOnly = 0x20,                      /* 1 - device is read only            */
  Removable = 0x10,                     /* 1 - device is removable            */
  ConsoleIn = 0x08,
  ConsoleOut = 0x04,
  Input = 0x02,
  Output = 0x01
  } DEVICE_FLAGS;

typedef enum _BUS_ID {
  ISADEVICE = 0x01,
  EISADEVICE = 0x02,
  PCIDEVICE = 0x04,
  PCMCIADEVICE = 0x08,
  PNPISADEVICE = 0x10,
  MCADEVICE = 0x20,
  MXDEVICE = 0x40,                      /* Devices on mezzanine bus           */
  PROCESSORDEVICE = 0x80,               /* Devices on processor bus           */
  VMEDEVICE = 0x100,
  } BUS_ID;
```

```
typedef struct _DEVICE_ID {
  unsigned long BusId;                  /* See BUS_ID enum above             */
  unsigned long DevId;                  /* Big Endian format                 */
  unsigned long SerialNum;              /* For multiple usage of a single
                                           DevId                             */
  unsigned long Flags;                  /* See DEVICE_FLAGS enum above       */
  unsigned char BaseType;               /* See pnp.h for bit definitions     */
  unsigned char SubType;                /* See pnp.h for bit definitions     */
  unsigned char Interface;              /* See pnp.h for bit definitions     */
  unsigned char Spare;
  } DEVICE_ID;

typedef union _BUS_ACCESS {
  struct _PnPAccess{
    unsigned char CSN;
    unsigned char LogicalDevNumber;
    unsigned short ReadDataPort;
    } PnPAccess;
  struct _ISAAccess{
    unsigned char SlotNumber;           /* ISA Slot Number generally not
                                           available; 0 if unknown           */
    unsigned char LogicalDevNumber;
    unsigned short ISAReserved;
    } ISAAccess;
  struct _MCAAccess{
    unsigned char SlotNumber;
    unsigned char LogicalDevNumber;
    unsigned short MCAReserved;
    } MCAAccess;
  struct _PCMCIAAccess{
    unsigned char SlotNumber;
    unsigned char LogicalDevNumber;
    unsigned short PCMCIAReserved;
    } PCMCIAAccess;
  struct _EISAAccess{
    unsigned char SlotNumber;
    unsigned char FunctionNumber;
    unsigned short EISAReserved;
    } EISAAccess;
  struct _PCIAccess{
    unsigned char BusNumber;
    unsigned char DevFuncNumber;
    unsigned short PCIReserved;
    } PCIAccess;
  struct _ProcBusAccess{
    unsigned char BusNumber;
    unsigned char BUID;
    unsigned short ProcBusReserved;
    } ProcBusAccess;
  } BUS_ACCESS;

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

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

typedef enum _CPU_STATE {
```

```
  CPU_GOOD = 0,                     /* CPU is present, and active         */
  CPU_GOOD_FW = 1,                  /* CPU is present, and in firmware    */
  CPU_OFF = 2,                      /* CPU is present, but inactive       */
  CPU_FAILED = 3                    /* CPU is present, but failed POST    */
  CPU_NOT_PRESENT = 255             /* CPU not present                    */
  } CPU_STATE;

typedef struct _PPC_CPU {
  unsigned long CpuType;            /* Result of mfspr from Processor
                                       Version Register (PVR).
                                       PVR(0-15) = Version (e.g. 601)
                                       PVR(16-31 = EC Level               */
  unsigned char CpuNumber;          /* CPU Number for this processor      */
  unsigned char CpuState;           /* CPU State, see CPU_STATE enum      */
  unsigned short Reserved;
  } PPC_CPU;

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

typedef enum _MEM_USAGE {
  Other = 0x8000,
  ResumeBlock = 0x4000,             /* for use by power management        */
  SystemROM = 0x2000,               /* Flash memory (populated)           */
  UnPopSystemROM = 0x1000,          /* Unpopulated part of SystemROM area */
  IOMemory = 0x0800,
  SystemIO = 0x0400,
  SystemRegs = 0x0200,
  PCIAddr = 0x0100,
  PCIConfig = 0x80,
  ISAAddr = 0x40,
  Unpopulated = 0x20,               /* Unpopulated part of System Memory  */
  Free = 0x10,                      /* Free part of System Memory         */
  BootImage = 0x08,                 /* BootImage part of System Memory    */
  FirmwareCode = 0x04,              /* FirmwareCode part of System Memory */
  FirmwareHeap = 0x02,              /* FirmwareHeap part of System Memory */
  FirmwareStack = 0x01              /* FirmwareStack part of System Memory*/
  } MEM_USAGE;

typedef struct _MEM_MAP {
  unsigned long Usage;              /* See MEM_USAGE above                */
  unsigned long BasePage;           /* Page number measured in 4KB pages  */
  unsigned long PageCount;          /* Page count measured in 4KB pages   */
  } MEM_MAP;

typedef struct _RESIDUAL {
  unsigned long ResidualLength;     /* Length of Residual                 */
  unsigned char Version;            /* of this data structure             */
  unsigned char Revision;           /* of this data structure             */
  unsigned short EC;                /* of this data structure             */
  /* VPD */
  VPD VitalProductData;
  /* CPU */
  unsigned short MaxNumCpus;        /* Max CPUs in this system            */
  unsigned short ActualNumCpus;     /* ActualNumCpus < MaxNumCpus means   */
                                    /* that there are unpopulated or      */
                                    /* otherwise unusable cpu locations   */
  PPC_CPU Cpus[MAX_CPUS];
  /* Memory */
  unsigned long TotalMemory;        /* Total amount of memory installed   */
  unsigned long GoodMemory;         /* Total amount of good memory        */
  unsigned long ActualNumMemSegs;
```

```
  MEM_MAP Segs[MAX_MEM_SEGS];
  unsigned long ActualNumMemories;
  PPC_MEM Memories[MAX_MEMS];
  /* Devices */
  unsigned long ActualNumDevices;
  PPC_DEVICE Devices[MAX_DEVICES];
  unsigned char DevicePnPHeap[2*MAX_DEVICES*AVE_PNP_SIZE];
  } RESIDUAL;

#endif  /* ndef _RESIDUAL_ */
```

pnp.h Appendix B

```
/* 5/18/95                                                                    */
/*----------------------------------------------------------------------------*/
/*      Plug and Play header definitions                                      */
/*----------------------------------------------------------------------------*/

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

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

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

#ifndef _PNP_
#define _PNP_

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

/* Interrupt controllers */

#define PNPinterrupt0 "PNP0000"      /* AT Interrupt Controller               */
#define PNPinterrupt1 "PNP0001"      /* EISA Interrupt Controller             */
#define PNPinterrupt2 "PNP0002"      /* MCA Interrupt Controller              */
#define PNPinterrupt3 "PNP0003"      /* APIC                                  */
#define PNPExtInt     "IBM000D"      /* PowerPC Extended Interrupt Controller */

/* Timers */

#define PNPtimer0     "PNP0100"      /* AT Timer                              */
#define PNPtimer1     "PNP0101"      /* EISA Timer                            */
#define PNPtimer2     "PNP0102"      /* MCA Timer                             */

/* DMA controllers */

#define PNPdma0       "PNP0200"      /* AT DMA Controller                     */
#define PNPdma1       "PNP0201"      /* EISA DMA Controller                   */
#define PNPdma2       "PNP0202"      /* MCA DMA Controller                    */

/* start of August 15, 1994 additions */
/* CMOS */
#define PNPCMOS       "IBM0009"      /* CMOS                                  */
```

```
/* L2 Cache */
#define PNPL2          "IBM0007"      /* L2 Cache                              */

/* NVRAM */
#define PNPNVRAM       "IBM0008"      /* NVRAM                                 */

/* Power Management */
#define PNPPM          "IBM0005"      /* Power Management                      */
/* end of August 15, 1994 additions */

/* Keyboards */

#define PNPkeyboard0  "PNP0300"       /* IBM PC/XT KB Cntlr (83 key, no mouse) */
#define PNPkeyboard1  "PNP0301"       /* Olivetti ICO (102 key)                */
#define PNPkeyboard2  "PNP0302"       /* IBM PC/AT KB Cntlr (84 key)           */
#define PNPkeyboard3  "PNP0303"       /* IBM Enhanced (101/2 key, PS/2 mouse)  */
#define PNPkeyboard4  "PNP0304"       /* Nokia 1050 KB Cntlr                   */
#define PNPkeyboard5  "PNP0305"       /* Nokia 9140 KB Cntlr                   */
#define PNPkeyboard6  "PNP0306"       /* Standard Japanese KB Cntlr            */
#define PNPkeyboard7  "PNP0307"       /* Microsoft Windows (R) KB Cntlr        */

/* Parallel port controllers */

#define PNPparallel0 "PNP0400"        /* Standard LPT Parallel Port            */
#define PNPparallel1 "PNP0401"        /* ECP Parallel Port                     */
#define PNPepp       "IBM001C"        /* EPP Parallel Port                     */

/* Serial port controllers */

#define PNPserial0   "PNP0500"        /* Standard PC Serial port               */
#define PNPSerial1   "PNP0501"        /* 16550A Compatible Serial port         */

/* Disk controllers */

#define PNPdisk0     "PNP0600"        /* Generic ESDI/IDE/ATA Compat HD Cntlr  */
#define PNPdisk1     "PNP0601"        /* Plus Hardcard II                      */
#define PNPdisk2     "PNP0602"        /* Plus Hardcard IIXL/EZ                 */

/* Diskette controllers */

#define PNPdiskette0 "PNP0700"        /* PC Standard Floppy Disk Controller    */

/* Display controllers */

#define PNPdisplay0  "PNP0900"        /* VGA Compatible                        */
#define PNPdisplay1  "PNP0901"        /* Video Seven VGA                       */
#define PNPdisplay2  "PNP0902"        /* 8514/A Compatible                     */
#define PNPdisplay3  "PNP0903"        /* Trident VGA                           */
#define PNPdisplay4  "PNP0904"        /* Cirrus Logic Laptop VGA               */
#define PNPdisplay5  "PNP0905"        /* Cirrus Logic VGA                      */
#define PNPdisplay6  "PNP0906"        /* Tseng ET4000 or ET4000/W32            */
#define PNPdisplay7  "PNP0907"        /* Western Digital VGA                   */
#define PNPdisplay8  "PNP0908"        /* Western Digital Laptop VGA            */
#define PNPdisplay9  "PNP0909"        /* S3                                    */
#define PNPdisplayA  "PNP090A"        /* ATI Ultra Pro/Plus (Mach 32)          */
#define PNPdisplayB  "PNP090B"        /* ATI Ultra (Mach 8)                    */
#define PNPdisplayC  "PNP090C"        /* XGA Compatible                        */
#define PNPdisplayD  "PNP090D"        /* ATI VGA Wonder                        */
#define PNPdisplayE  "PNP090E"        /* Weitek P9000 Graphics Adapter         */
#define PNPdisplayF  "PNP090F"        /* Oak Technology VGA                    */

/* Peripheral busses */

#define PNPbuses0    "PNP0A00"        /* ISA Bus                               */
```

```
#define PNPbuses1     "PNP0A01"       /* EISA Bus                              */
#define PNPbuses2     "PNP0A02"       /* MCA Bus                               */
#define PNPbuses3     "PNP0A03"       /* PCI Bus                               */
#define PNPbuses4     "PNP0A04"       /* VESA/VL Bus                           */

/* RTC, BIOS, planar devices */

#define PNPspeaker0   "PNP0800"       /* AT Style Speaker Sound                */
#define PNPrtc0       "PNP0B00"       /* AT RTC                                */
#define PNPpnpbios0   "PNP0C00"       /* PNP BIOS (only created by root enum)  */
#define PNPpnpbios1   "PNP0C01"       /* System Board Memory Device            */
#define PNPpnpbios2   "PNP0C02"       /* Math Coprocessor                      */
#define PNPpnpbios3   "PNP0C03"       /* PNP BIOS Event Notification Interrupt */

/* PCMCIA controller */

#define PNPpcmcia0    "PNP0E00"       /* Intel 82365 Compatible PCMCIA Cntlr   */

/* Mice */

#define PNPmouse0     "PNP0F00"       /* Microsoft Bus Mouse                   */
#define PNPmouse1     "PNP0F01"       /* Microsoft Serial Mouse                */
#define PNPmouse2     "PNP0F02"       /* Microsoft Inport Mouse                */
#define PNPmouse3     "PNP0F03"       /* Microsoft PS/2 Mouse                  */
#define PNPmouse4     "PNP0F04"       /* Mousesystems Mouse                    */
#define PNPmouse5     "PNP0F05"       /* Mousesystems 3 Button Mouse - COM2    */
#define PNPmouse6     "PNP0F06"       /* Genius Mouse - COM1                   */
#define PNPmouse7     "PNP0F07"       /* Genius Mouse - COM2                   */
#define PNPmouse8     "PNP0F08"       /* Logitech Serial Mouse                 */
#define PNPmouse9     "PNP0F09"       /* Microsoft Ballpoint Serial Mouse      */
#define PNPmouseA     "PNP0F0A"       /* Microsoft PNP Mouse                   */
#define PNPmouseB     "PNP0F0B"       /* Microsoft PNP Ballpoint Mouse         */

/* Modems */

#define PNPmodem0     "PNP9000"       /* Specific IDs TBD                      */

/* Network controllers */

#define PNPnetworkC9 "PNP80C9"        /* IBM Token Ring                        */
#define PNPnetworkCA "PNP80CA"        /* IBM Token Ring II                     */
#define PNPnetworkCB "PNP80CB"        /* IBM Token Ring II/Short               */
#define PNPnetworkCC "PNP80CC"        /* IBM Token Ring 4/16Mbs                */
#define PNPnetwork27 "PNP8327"        /* IBM Token Ring (All types)            */
#define PNPnetworket "IBM0010"        /* IBM Ethernet used by Power PC         */
#define PNPneteisaet "IBM2001"        /* IBM Ethernet EISA adapter             */
#define PNPAMD79C970 "IBM0016"        /* AMD 79C970 (PCI Ethernet)             */

/* SCSI controllers */

#define PNPscsi0      "PNPA000"       /* Adaptec 154x Compatible SCSI Cntlr    */
#define PNPscsi1      "PNPA001"       /* Adaptec 174x Compatible SCSI Cntlr    */
#define PNPscsi2      "PNPA002"       /* Future Domain 16-700 Compat SCSI Cntlr*/
#define PNPscsi3      "PNPA003"       /* Panasonic CDROM Adapter (SBPro/SB16)  */
#define PNPscsiF      "IBM000F"       /* NCR 810 SCSI Controller               */
#define PNPscsi825    "IBM001B"       /* NCR 825 SCSI Controller               */

/* Sound/Video, Multimedia */

#define PNPmm0        "PNPB000"       /* Sound Blaster Compatible Sound Device */
#define PNPmm1        "PNPB001"       /* MS Windows Sound System Compat Device */
#define PNPmmF        "IBM000E"       /* Crystal CS4231 Audio Device           */
#define PNPv7310      "IBM0015"       /* ASCII V7310 Video Capture Device      */
#define PNPmm4232     "IBM0017"       /* Crystal CS4232 Audio Device           */
```

```
#define PNPpmsyn     "IBM001D"       /* YMF 289B chip (Yamaha)               */
#define PNPgp4232    "IBM0012"       /* Crystal CS4232 Game Port             */
#define PNPmidi4232  "IBM0013"       /* Crystal CS4232 MIDI                  */

/* Operator Panel */
#define PNPopctl     "IBM000B"       /* Operator's panel                     */

/* Service Processor */
#define PNPsp        "IBM0011"       /* IBM Service Processor                */

/* Memory Controller */
#define PNPmemctl    "IBM000A"       /* Memory controller                    */

/* Graphics Assist */
#define PNPg_assist  "IBM0014"       /* Graphics Assist                      */

/* PNP Packet Handles */

#define S1_Packet                0x0A   /* Version resource                   */
#define S2_Packet                0x15   /* Logical DEVID (without flags)      */
#define S2_Packet_flags          0x16   /* Logical DEVID (with flags)         */
#define S3_Packet                0x1C   /* Compatible device ID               */
#define S4_Packet                0x22   /* IRQ resource (without flags)       */
#define S4_Packet_flags          0x23   /* IRQ resource (with flags)          */
#define S5_Packet                0x2A   /* DMA resource                       */
#define S6_Packet                0x30   /* Depend funct start (w/o priority)  */
#define S6_Packet_priority       0x31   /* Depend funct start (w/ priority)   */
#define S7_Packet                0x38   /* Depend funct end                   */
#define S8_Packet                0x47   /* I/O port resource (w/o fixed loc)  */
#define S9_Packet_fixed          0x4B   /* I/O port resource (w/ fixed loc)   */
#define S14_Packet               0x71   /* Vendor defined                     */
#define S15_Packet               0x78   /* End of resource (w/o checksum)     */
#define S15_Packet_checksum      0x79   /* End of resource (w/ checksum)      */
#define L1_Packet                0x81   /* Memory range                       */
#define L1_Shadow                0x20   /* Memory is shadowable               */
#define L1_32bit_mem             0x18   /* 32-bit memory only                 */
#define L1_8_16bit_mem           0x10   /* 8- and 16-bit supported            */
#define L1_Decode_Hi             0x04   /* decode supports high address       */
#define L1_Cache                 0x02   /* read cacheable, write-through      */
#define L1_Writeable             0x01   /* Memory is writeable                */
#define L2_Packet                0x82   /* ANSI ID string                     */
#define L3_Packet                0x83   /* Unicode ID string                  */
#define L4_Packet                0x84   /* Vendor defined                     */
#define L5_Packet                0x85   /* Large I/O                          */
#define L6_Packet                0x86   /* 32-bit Fixed Loc Mem Range Desc    */
#define END_TAG                  0x78   /* End of resource                    */
#define DF_START_TAG             0x30   /* Dependent function start           */
#define DF_START_TAG_priority    0x31   /* Dependent function start           */
#define DF_END_TAG               0x38   /* Dependent function end             */
#define SUBOPTIMAL_CONFIGURATION 0x2    /* Priority byte sub optimal config   */

/* Device Base Type Codes */

typedef enum _PnP_BASE_TYPE {
  Reserved = 0,
  MassStorageDevice = 1,
  NetworkInterfaceController = 2,
  DisplayController = 3,
  MultimediaController = 4,
  MemoryController = 5,
  BridgeController = 6,
  CommunicationsDevice = 7,
  SystemPeripheral = 8,
  InputDevice = 9,
```

```
  } PnP_BASE_TYPE;

/* Device Sub Type Codes */

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

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

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

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

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

  HostProcessorBridge = 0,
  ISABridge = 1,
  EISABridge = 2,
  MicroChannelBridge = 3,
  PCIBridge = 4,
  PCMCIABridge = 5,
  VMEBridge = 6,
  OtherBridgeDevice = 0x80,

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

  ProgrammableInterruptController = 0,
  DMAController = 1,
  SystemTimer = 2,
  RealTimeClock = 3,
  L2Cache = 4,
  NVRAM = 5,
  PowerManagement = 6,
  CMOS = 7,
  OperatorPanel = 8,
  ServiceProcessorClass1 = 9,
  ServiceProcessorClass2 = 0xA,
  ServiceProcessorClass3 = 0xB,
  GraphicAssist = 0xC,
  SystemPlanar = 0xF,
  OtherSystemPeripheral = 0x80,

  KeyboardController = 0,
  Digitizer = 1,
  MouseController = 2,
  OtherInputController = 0x80,

  GeneralMemoryController = 0,
```

```
  } PnP_SUB_TYPE;

/* Device Interface Type Codes */

typedef enum _PnP_INTERFACE {
  General = 0,
  GeneralSCSI = 0,
  GeneralIDE = 0,
  ATACompatible = 1,

  GeneralFloppy = 0,
  Compatible765 = 1,
  NS398_Floppy = 2,                    /* NS Super I/O wired to use index
                                          register at port 398 and data
                                          register at port 399              */
  NS26E_Floppy = 3,                    /* Ports 26E and 26F                 */
  NS15C_Floppy = 4,                    /* Ports 15C and 15D                 */
  NS2E_Floppy = 5,                     /* Ports 2E and 2F                   */
  CHRP_Floppy = 6,                     /* CHRP Floppy in PR*P Systems       */

  GeneralIPI = 0,

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

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

  GeneralVideo = 0,
  GeneralAudio = 0,
  CS4232Audio = 1,                     /* CS 4232 Plug 'n Play Configured   */

  GeneralRAM = 0,
  GeneralFLASH = 0,
  PCIMemoryController = 0,             /* PCI Config Method                 */
  RS6KMemoryController = 1,            /* RS6K Config Method                */

  GeneralHostBridge = 0,
  GeneralISABridge = 0,
  GeneralEISABridge = 0,
  GeneralMCABridge = 0,
  GeneralPCIBridge = 0,
  PCIBridgeDirect = 0,
  PCIBridgeIndirect = 1,
  PCIBridgeRS6K = 2,
  GeneralPCMCIABridge = 0,
  GeneralVMEBridge = 0,

  GeneralRS232 = 0,
  COMx = 1,
  Compatible16450 = 2,
  Compatible16550 = 3,
  NS398SerPort = 4,                    /* NS Super I/O wired to use index
                                          register at port 398 and data
                                          register at port 399              */
  NS26ESerPort = 5,                    /* Ports 26E and 26F                 */
  NS15CSerPort = 6,                    /* Ports 15C and 15D                 */
  NS2ESerPort = 7,                     /* Ports 2E and 2F                   */

  GeneralParPort = 0,
  LPTx = 1,
  NS398ParPort = 2,                    /* NS Super I/O wired to use index
```

```
                                        register at port 398 and data
                                        register at port 399              */
  NS26EParPort = 3,                    /* Ports 26E and 26F                */
  NS15CParPort = 4,                    /* Ports 15C and 15D                */
  NS2EParPort = 5,                     /* Ports 2E and 2F                  */

  GeneralPIC = 0,
  ISA_PIC = 1,
  EISA_PIC = 2,
  MPIC = 3,
  RS6K_PIC = 4,

  GeneralDMA = 0,
  ISA_DMA = 1,
  EISA_DMA = 2,

  GeneralTimer = 0,
  ISA_Timer = 1,
  EISA_Timer = 2,
  GeneralRTC = 0,
  ISA_RTC = 1,

  StoreThruOnly = 1,
  StoreInEnabled = 2,
  RS6KL2Cache = 3,

  IndirectNVRAM = 0,                   /* Indirectly addressed             */
  DirectNVRAM = 1,                     /* Memory Mapped                    */
  IndirectNVRAM24 = 2,                 /* Indirectly addressed - 24 bit    */

  GeneralPowerManagement = 0,
  EPOWPowerManagement = 1,
  PowerControl = 2,                   // d1378

  GeneralCMOS = 0,

  GeneralOPPanel = 0,
  HarddiskLight = 1,
  CDROMLight = 2,
  PowerLight = 3,
  KeyLock = 4,
  ANDisplay = 5,                       /* AlphaNumeric Display             */
  SystemStatusLED = 6,                 /* 3 digit 7 segment LED            */
  CHRPSystemStatusLED = 7,             /* CHRP LEDs in PR*P System         */

  GeneralServiceProcessor = 0,

  TransferData = 1,
  IGMC32 = 2,
  IGMC64 = 3,

  GeneralSystemPlanar = 0,

  } PnP_INTERFACE;

/* PnP resources */

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

typedef struct _SERIAL_ID {
  unsigned char VendorID0;             /*    Bit(7)=0                      */
                                       /*    Bits(6:2)=1st character in    */
                                       /*       compressed ASCII           */
                                       /*    Bits(1:0)=2nd character in    */
```

```
                                        /*       compressed ASCII bits(4:3)   */
  unsigned char VendorID1;              /*    Bits(7:5)=2nd character in      */
                                        /*       compressed ASCII bits(2:0)   */
                                        /*    Bits(4:0)=3rd character in      */
                                        /*       compressed ASCII             */
  unsigned char VendorID2;              /* Product number - vendor assigned   */
  unsigned char VendorID3;              /* Product number - vendor assigned   */

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

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

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

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

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

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

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

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

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

typedef union _PnP_TAG_PACKET {
  struct _S1_Pack{                      /* VERSION PACKET                     */
```

```
    unsigned char Tag;                  /* small tag = 0x0a                    */
    unsigned char Version[2];           /* PnP version, Vendor version         */
    } S1_Pack;

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

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

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

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

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

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

  struct _S8_Pack{                      /* VARIABLE I/O PORT PACKET            */
    unsigned char Tag;                  /* small tag x47                       */
    unsigned char IOInfo;               /* x0  = decode only bits(9:0);        */
#define  ISAAddr16bit         0x01      /* x01 = decode bits(15:0)             */
    unsigned char RangeMin[2];          /* Min base address                    */
    unsigned char RangeMax[2];          /* Max base address                    */
    unsigned char IOAlign;              /* base alignmt, incr in 1B blocks     */
    unsigned char IONum;                /* number of contiguous I/O ports      */
    } S8_Pack;

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

  struct _S14_Pack{                     /* VENDOR DEFINED PACKET               */
```

```
   unsigned char Tag;                 /* small tag = 0x7m m = 1-7          */
   union _S14_Data{
     unsigned char Data[7];           /* Vendor defined                    */
     struct _S14_PPCPack{             /* Pr*p s14 pack                     */
        unsigned char Type;           /* 00=non-IBM                        */
        unsigned char PPCData[6];     /* Vendor defined                    */
       } S14_PPCPack;
     } S14_Data;
   } S14_Pack;

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

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

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

 struct _L3_Pack{                    /* UNICODE ID STRING PACKET          */
   unsigned char Tag;                /* large tag = 0x83                  */
   unsigned char Count0;             /* Length + 2 of string              */
   unsigned char Count1;
   unsigned char Country0;           /* TBD                               */
   unsigned char Country1;           /* TBD                               */
   unsigned char Identifier[1];      /* a variable array of bytes,        */
                                     /* count in tag                      */
   } L3_Pack;

 struct _L4_Pack{                    /* VENDOR DEFINED PACKET             */
   unsigned char Tag;                /* large tag = 0x84                  */
   unsigned char Count0;
   unsigned char Count1;
   union _L4_Data{
     unsigned char Data[1];          /* a variable array of bytes,        */
                                     /* count in tag                      */
     struct _L4_PPCPack{             /* Pr*p L4 packet                    */
        unsigned char Type;          /* 00=non-IBM                        */
        unsigned char PPCData[1];    /* a variable array of bytes,        */
                                     /* count in tag                      */
       } L4_PPCPack;
     } L4_Data;
   } L4_Pack;

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

 struct _L6_Pack{
```

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

  } PnP_TAG_PACKET;

#endif  /* ndef _PNP_ */
```