fined, and the firmware must not try to return control to the client program or reboot the system. The suggested behavior in this case is to enter a firmware-interactive mode, if available.

The firmware may recognize strings other than “power-off” in a system-dependent manner.

If the halt string is not a recognized command, the firmware must reboot the system. During the system reset and the firmware restart, the firmware must preserve the halt string. Then, the firmware must evaluate the string as if “auto-boot?” were true and “boot-command” were set to that string, without altering the values of the “auto-boot?” and “boot-command” Configuration Variables. If the firmware includes an Open Firmware user interface, the string can be any valid user interface command string. Otherwise, the firmware must interpret the string as shown in Table 37.

Table 37. Halt Arguments

String Value	Meaning
boot	Load and execute the default client program
<empty string>*	If the firmware has an interactive mode, enter that mode. Otherwise the result is undefined, except that the firmware must not perform the “boot” action.
<any other string>	Behavior is system-dependent

Note: *<empty string> means a valid address that points to a null character. An address value of 0 is not an empty string.

I.9 User Interface Requirements

An implementation of PowerPC Open Firmware must conform to the core requirements as specified in [1] and the following PowerPC specific extensions.

I.9.1 Machine Register Access

The following user interface commands represent PowerPC registers within the saved program state. Executing the command returns the saved value of the corresponding register. The saved value may be set by preceding the command with **to**; the actual registers are restored to the saved values when **go** is executed.

I.9.1.1 Branch Unit Registers

%cr Access saved copy of Condition Register.

%ctr Access saved copy of Count Register.

%lr Access saved copy of Link Register.

%msr Access saved copy of Machine State Register.

%srr0 and %srr1 Access saved copy of Save/Restore Registers.

I.9.1.2 Fixed-Point Registers

%r0 through %r31 Access saved copies of Fixed-Point Registers.

%xer Access saved copy of XER Register.

%sprg0 through %sprg3 Access saved copies of SPRG registers.

I.9.1.3 Floating-Point Registers

Unlike the other registers, the floating-point unit registers are not normally saved, since they are not used by Open Firmware. The following access words, therefore, access the registers directly.

%f0 through %f31	Access Floating-Point Registers
%fpscr	Access Floating-Point Status and Control Register

The following command displays the PowerPC CPU saved program state.

.registers

I.10 Configuration Variables

In addition to the standard Configuration Variables defined by the core Open Firmware document [1], the following Configuration Variables must be implemented for PowerPC Open Firmware:

"little-endian?"	This Boolean variable controls the Endian mode of Open Firmware. If **true** (-1), the Endian mode is Big-Endian; if **false** (0), the Endian mode is Little-Endian. The default value is implementation dependent.
"real-mode?"	This Boolean variable controls the address translation mode of Open Firmware. If **true** (-1), the addressing mode is Real-Mode; if **false** (0), the addressing mode is Virtual-Mode. The default value is implementation dependent.
"my-base"	This integer variable defines the starting address to be used by Open Firmware. When Open Firmware is running in Real-Mode, **"my-base"** must be the physical memory address.
"my-size"	This integer variable defines the size of the address space which is used by Open Firmware. When Open Firmware is running in Real-Mode, **"my-size"** must be the physical memory size.
"load-base"	This integer variable defines the default load address for client programs when using the **load** method. The default value is implementation dependent.

I.11 Terminal Emulator Support Package

IEEE Std 1275-1994 defines the behavior of the Terminal Emulator Support Package (see Annex B of [1]). Annex B of [1] assumes that the Terminal Emulator Support Package implements certain escape sequences from the set defined by ANSI X3.64. The extension described here corresponds to ISO 6429-1983, as defined by [9]. An implementation of PowerPC Open Firmware is required to support additional graphic renditions (via SGR (Select Graphic Rendition) -- **ESC**[**#m**) beyond those specified in Annex B of [1].

In order for these extensions to be used, the FCode device driver for a display device (i.e. a device whose *"device_type"* property has the value *"display"*) must initialize the first 16 entries of its color table to appropriate values (see Section I.12.1.1, "Color Table Initialization"); note that this setup is standard for "VGA"-type devices. These values assume that the color is represented by the three LSbs of the color index and that the fourth LSb which corresponds to a value of 8 represents the intensity. The ISO 6429-1983 standard provides parameter values to independently control the color of foreground (30-37) and background (40-47). The intensity is set separately (1-2), and must be issued before the color control.

The expanded model of the Terminal Emulator is that there is a "current" background and foreground color (index), each of whose value is 0-15, corresponding to the first 16 entries of the color table. In positive

image mode, pixels corresponding to a font (logo) bit set (1) must be set to the foreground color; pixels corresponding to a font (logo) bit clear (0) must be set to the background color. When in negative image mode, the roles of foreground and background are reversed.

The default rendition must be positive image mode, background=15 and foreground=0, thus producing black characters on a bright white background.

The following table describes the effect of executing the SGR escape sequence with the specified parameter.

Table 38. SGR Parameters

Parameter	**Interpretation**
0	Default rendition
1	Bold (increased intensity)
2	Faint (decreased intensity)
7	Negative image
27	Positive image
30	Black foreground
31	Red foreground
32	Green foreground
33	Yellow foreground
34	Blue foreground
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black background
41	Red background
42	Green background
43	Yellow background
44	Blue background
45	Magenta background
46	Cyan background
47	White background

I.11.1 Display Device Low-Level Interfaces

PowerPC Open Firmware must implement the following method in the terminal emulation support package.

draw-logo-in-color (line# addr width height --) Draw (at line#) the logo stored at location *addr*

This method implements 8-bit-per-pixel color drawing. The FCode device driver for a display device must initialize the first 16 entries of its color table to the appropriate values as defined in Section I.12.1.1, “Color Table Initialization.”

In addition to the methods defined in Section 3.8.4.3 of [1], execution of **is-install** creates the **draw-logo-in-color** method in the current package that will execute the draw-logo-in-color deferred word.

When the *"fb8"* generic frame buffer package implements the display device low-level interface for a frame buffer, execution of **fb8-install** must install the **fb8-draw-logo-in-color** routine as the behavior of the draw-logo-in-color deferred word.

fb8-draw-logo-in-color (line# addr width height --) Implement the "fb8" draw-logo-in-color

I.12 Extensions for PowerPC Based Systems

This section describes the properties, methods and device subtrees that are applicable to devices required by the PowerPC Reference Platform system architecture. It is strongly recommended that other platforms follow these definitions for the corresponding devices.

I.12.1 Display Devices

This section defines additional behavior of display devices (e.g. *device_type = "display"*) for PowerPC Reference Platform Open Firmware implementations.

I.12.1.1 Color Table Initialization

The *PowerPC Reference Platform Specification* requires that display devices support a minimum of 256 colors. The core specification of Open Firmware defines a Terminal Emulation Support Package that, while defined for 8-bit pixels, does not include support for colors. Open Firmware implementations for the PowerPC Reference Platform must support additional Select Graphic Rendition parameters (see Section I.11, "Terminal Emulator Support Package") in order to allow client programs to display characters (and logo) using a 16-color model.

For this expanded Terminal Emulation support to work, Open Firmware device drivers for *"display"* devices must initialize the first 16 entries of their color table to values defined in Table 39. Note that the table values are defined in terms of percentage of full saturation color for each of the primary RGB colors.

Table 39 (Page 1 of 2). Color Table Values

Index	Red	Blue	Green	Color
0	0	0	0	Black
1	0	0	2/3	Blue
2	0	2/3	0	Green
3	0	2/3	2/3	Cyan
4	2/3	0	0	Red
5	2/3	0	2/3	Magenta
6	2/3	1/3	0	Brown
7	2/3	2/3	2/3	White
8	1/3	1/3	1/3	Gray
9	1/3	1/3	1	Light Blue
10	1/3	1	1/3	Light Green
11	1/3	1	1	Light Cyan

Table 39 (Page 2 of 2). Color Table Values				
Index	**Red**	**Blue**	**Green**	**Color**
12	1	1/3	1/3	Light Red
13	1	1/3	1	Light Magenta
14	1	1	1/3	Yellow
15	1	1	1	Bright White

I.12.1.2 "Display" Device Standard Properties

In addition to the standard properties defined by Open Firmware for display devices, the following properties must be present for PowerPC Reference Platform-compliant implementations.

"width" Standard property, encoded as with **encode-int**, that represents the visible width of the display in pixels.

"height" Standard property, encoded as with **encode-int**, that represents the visible height of the display in pixels.

"linebytes" Standard property, encoded as with **encode-int**, that represents the address offset between a pixel on one scan line and that same relative horizontal pixel position on the immediately following scan line.

"depth" Standard property, encoded as with **encode-int**, that represents the number of bits in each pixel.

I.12.2 Keyboard Devices

Open Firmware does not have a specific device class for keyboards; instead, keyboards are instances of the *"serial"* device class. However, certain features of keyboards (i.e. the ability to re-map keys, etc.) make it desirable to implement them as a separate device class.

In general, keyboard devices produce a hardware scan-code that is specific to the type of keyboard. These scan-codes are then mapped via software to produce the character code for the key, taking into account the state of "modifier" keys (e.g. Shift, Control) and the keyboard layout. The mapping of scan-codes to character value depends upon the keyboard layout. This is dependent upon the language that is being supported by the keyboard; e.g. the layout of keys for a French keyboard is different than that for an English keyboard.

For purposes of localization, it is necessary for the scan-code conversion to be controlled by the software, including Open Firmware (via the client interface). This section adds mechanisms to allow client programs to alter the scan-code conversion, based upon the keyboard (language) layout.

The language-specific layout is specified by means of a two-character abbreviation, as described by [11]. The following languages may be supported:

CS Czech

DA Danish

NL Dutch

EN English (default)

FI Finnish

FR French

| DE German

| HU Hungarian

| IT Italian

| NO Norwegian

| PL Polish

| PT Portuguese

| SK Slovak

| ES Spanish

| SV Swedish

| For testing of keyboard devices, additional low-level methods are defined for *"keyboard"* devices. **get-scancode** will read the next available scan-code from the keyboard; **scancode->char** will perform character mapping, using the same conversion as would normal **read**s. This enables test software to test for a specific character to terminate the test without having to know the scancode-to-character translations.

I.12.2.1 "Keyboard" Device Standard Properties

"device_type" Standard Open Firmware property; the value of this property for keyboard devices must be *"keyboard"*.

"language" Standard property, encoded as with **encode-string**, that indicates the current scan-code-to-character conversion based upon the language's keyboard layout.

| The *"language"* standard property for keyboard devices is defined for PowerPC Open Firmware implementations. This value indicates the language (i.e. keyboard layout scan-code conversion) to which the keyboard driver is currently set.

I.12.2.2 "Keyboard" Device Methods

| In addition to the Open Firmware standard **open, close** and **read** methods, the following methods must be supported by an Open Firmware implementation of a *"keyboard"* device. Note that the **open** routine can take an optional argument which specifies the language (i.e. **scancode->char** mapping) to be used.

get-scancode **(msecs -- scancode true | false)**

This method returns the "raw" scan-code value of the next key alteration. *msecs* is the number of milliseconds to wait for a keystroke before reporting failure; a value of 0 implies wait until keystroke. If this timeout expires before a keystroke is read, **false** is returned. Otherwise, the *scancode* and **true** are returned.

scancode->char **(scancode -- char true | false)**

Using the *scancode*, perform translation to a character. If the *scancode* represents a modifier key (e.g. Shift), no translation will be available; in this case, **false** is returned. If the *scancode* represents a translated character, *char* and **true** are returned.

set-language **(str len -- true | false)**

If the keyboard driver can support the requested character set, it must set its *"language"* property to the value specified by *str,len*, and return a value of **true**. If it can not support the requested language, no change of *"language"* is made and a value of **false** is returned.

The reason for adding a special call, instead of just using the **property** (*setprop*) call, is to allow the device to "filter" the request; i.e. a scan-code conversion may not be available for the requested language. This call allows the device driver to change the property only if it can support the requested mapping.

I.12.3 Pointing Devices

This class of device covers a broad category of "pointing" devices, the most common embodiment of which is the mouse. These devices can typically generate X,Y coordinates and button-press information on some periodic basis. The following properties and methods are defined for such devices.

I.12.3.1 "Mouse" Device Standard Properties

"device_type" Standard Open Firmware property; the value of this property for pointing devices must be *"mouse"*.

"#buttons" Standard property, encoded as with **encode-int**, that indicates the number of physical buttons supported by the device. This property can be used to interpret the *buttons* value returned by the **get-event** method.

"absolute-position" Standard property, whose presence indicates that this device supplies absolute X,Y coordinates (e.g. a graphics tablet). Absence of this property indicates that the device supplies relative X,Y position (e.g. a mouse).

I.12.3.2 "Mouse" Device Methods

In addition to the Open Firmware standard **open** and **close** methods, the following methods must be supported by an Open Firmware implementation of a *"mouse"* device.

Pointing devices typically supply data only when an "event" occurs (e.g. the mouse moves or a button is pressed). The following method attempts to obtain an event from the device, reporting whether an event occurred.

get-event **(msecs -- pos.x pos.y buttons true | false)**

Standard method for obtaining the next "event" of pointing devices. *msecs* is the number of milliseconds to wait for an event before reporting failure; a value of 0 implies wait until event. *pos.x, pos.y* return the positioning information; they are interpreted as unsigned or signed, depending upon the presence or absence of the *"absolute-position"* property. *buttons* returns a bit-mask (in the low-order bits) representing any buttons that are pressed; the number of significant bits to examine is defined by the *"#buttons"* property. The least significant bit corresponds to the leftmost button of the device. The top stack result indicates whether an event was detected within the time-out period.

I.12.4 Real-Time Clock (RTC) Device

The PowerPC Reference Platform requires the presence of a Real-Time Clock, with a minimum resolution of one second. Open Firmware for this clock defines the following properties and methods. The representation of time is defined by the TIME&DATE method of the ANS Forth standard [10].

I.12.4.1 "RTC" Device Standard Properties

"device_type" Standard Open Firmware property; the value of this property must be *"rtc"*.

I.12.4.2 "RTC" Device Methods

In addition to the Open Firmware standard **open** and **close** methods, the following methods must be supported by an Open Firmware implementation of an *"rtc"* device.

get-time **(-- n1 n2 n3 n4 n5 n6)**

Return the current time as the integers *n1...n6*, where *n1* is the second {0...59}, *n2* is the minute {0...59}, *n3* is the hour {0...23}, *n4* is the day {1...31}, *n5* is the month {1...12}, and *n6* is the year (e.g. 1994).

set-time **(n1 n2 n3 n4 n5 n6 --)**

Set the current time from the integers *n1...n6*, where *n1* is the second {0...59}, *n2* is the minute {0...59}, *n3* is the hour {0...23}, *n4* is the day {1...31}, *n5* is the month {1...12}, and *n6* is the year (e.g. 1994).

I.12.5 Sound Device

The PowerPC Reference Platform requires an audio subsystem with at least two channels for input and two channels for output, capable of sampling rates of at least 22.05 and 44.1 KHz to at least 16-bit resolution. In order to use this sound device within the context of Open Firmware (e.g. "boot beeps"), the following properties and methods must be implemented.

I.12.5.1 "Sound" Device Standard Properties

"device-type" Standard Open Firmware property; the value of this property must be *"sound"*.

"#channels" Standard property, encoded as with **encode-int**, that represents the number of channels supported by the device.

"sample-resolution" Standard property, encoded as with **encode-int**, that represents the number of bits of resolution for each sound sample.

"sample-width" Standard property, encoded as with **encode-int**, that represents the number of bytes required for storing a sample.

"sample-rates" Standard property, consisting of an array of integers, each encoded as with **encode-int**, that represents the rates (in hertz) at which this device can be sampled.

I.12.5.2 "Sound" Device Standard Methods

The following methods must be implemented by a *"sound"* device.

open **(-- true | false)**

This Standard method prepares the *"sound"* device for subsequent **read**s or **write**s. An argument can be supplied (i.e. following a ":" in the path-name component, available via **my-args**) to specify sampling parameters. The argument is a string consisting of the external representation of the sample-rate to be used; if absent, an implementation-defined sample rate is used. An error is signalled by **open** (i.e. it returns **false**) if the requested sample rate cannot be supported by the device.

close **(--)**

Standard Open Firmware behavior.

read **(addr size -- actual)**

Acquire sound data, storing the samples at *addr*. The sample rate is established by **open**.

write **(addr size -- actual)**

Output sound samples, stored at *addr*. The sample rate is established by **open**.

I.12.6 NVRAM Device

Access to the PowerPC Reference Platform-specific area of NVRAM is supported by this device-type. The NVRAM is treated as a device that can be read and written using the standard Open Firmware **read** and **write** methods; the starting position within the NVRAM can be specified by the **seek** method.

The PowerPC Reference Platform standard defines CRCs that are used to validate the integrity of the data within the NVRAM. To provide flexibility in using NVRAM, options are provided on the **open** method for NVRAM that determine whether the CRCs have to be valid at **open** time, and/or whether they are written at **close** time. These options appear as the argument component of the device specifier used to **open** the device (as with **open-dev**).

I.12.6.1 NVRAM Properties

"device-type" Standard Open Firmware property; the value of this property must be *"nvram"*.

I.12.6.2 NVRAM Methods

The following methods have the semantics of the Open Firmware methods:

read **(addr len -- actual)**

write **(addr len -- actual)**

seek **(pos.lo pos.hi -- status)**

The following methods have additional behavior depending upon the argument used to open the device:

open **(-- true | false)**
Standard method used to initiate access to the device and control how CRCs are used and/or generated. The **open** method must use **my-args** to determine special handling, as follows:

- **(no argument)** — **my-args** is not provided. The **open** method will verify the CRCs within the PowerPC Reference Platform's NVRAM area. If correct, the **open** will succeed, returning **true**; subsequent calls to **write** will cause the CRCs to be calculated and stored when the device is **close**'d. If not correct, **open** will report failure by returning **false**.
- **raw** — The **open** will succeed, without verifying the CRCs. Calls to **write** will not be reflected in the CRCs at **close** time.
- **repair** — The **open** will succeed, without verifying the CRCs. However, the CRCs will be regenerated at **close** time.
- **close** **(--)** Standard method, whose behavior depends upon what the argument value was at the time the device was **open**'d. If *argument* was empty, or *repair*, the CRCs will be computed and stored.

I.12.7 Parallel Port Device

Access to the PowerPC Reference Platform parallel port is supported by this device type. The parallel port is treated as a byte-stream device.

I.12.7.1 Parallel Port Properties

"device-type" Standard Open Firmware property. The value of this property must be *"parallel"*.

I.12.7.2 Parallel Port Methods

The following methods have the semantics of the corresponding Open Firmware standard methods:

open **(-- true | false)**

close **(--)**

write **(addr len -- actual)**

I.12.8 Conventions for Devices on ISA and SCSI Buses

This section defines the naming and device type conventions for typical devices on ISA and SCSI buses. The following list shows the values of the "name" and "device-type" properties of the devices on an ISA bus:

name	**device_type**
8042	
kbd	"keyboard"
mouse	"mouse"
floppy	"block"
com	"serial"
timer	"timer"
lpt	"parallel"
ide	"block"
nvram	"nvram"
rtc	"rtc"

Note: The "kbd" and "mouse" names are indented to show that they are the child nodes of the 8042 node.

Some systems use an I/O controller, often called a super I/O chip, which provides control functions of multiple I/O devices. When a system uses a super I/O chip, the device node of the super I/O chip must not be created. Instead, the device nodes of the devices attached to the super I/O chip must be implemented as a child node of the bus node to which the super I/O chip is attached.

The following list shows the values of the name and device-type properties of the devices on a SCSI bus:

name	**device_type**
scsi	"scsi"
disk	"block"
tape	"byte"

Note: The SCSI controller is considered a bus device in the device tree for PowerPC Open Firmware. The "disk" and "tape" names are indented to show that they are the child nodes of the "scsi" node.

If there are multiple instances of the same device type, for example two IDE hard disks or two SCSI hard disks, their names must be postfixed with indices to distinguish them, such as ide1 and ide2, or disk1 and disk2.

It is strongly recommended that the "compatible" property be implemented for ISA and SCSI bus devices to help operating systems find appropriate device drivers for these devices.

I.12.9 "/aliases" Node Properties

An implementation of Open Firmware for the PowerPC Reference Platform must provide the following aliases (for the preferred devices that exist) if applicable device exists under the "/aliases" node:

```
| disk
| tape
| cdrom
| keyboard
| screen
| scsi
| com1
| com2
| floppy
| net
```

Appendix J. Plug and Play Extensions

The structure below is the Plug and Play definition for PCI adaptors.

```
/* Structure map for PCI Bridge in PnP Vendor specific packet */

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

#ifndef _PCIPNP_
#define _PCIPNP_

#define MAX_PCI_INTS    4

typedef enum _IntTypes {              /* interrupt controller types enumerator */
   IntCtl8259 = 1,                    /*    8259                               */
   IntCtlMPIC = 2                     /*    MPIC                               */
   } _IntTypes;

typedef struct _IntStruct {           /* PCI Int to IRQ conversion map */
   unsigned char  IntCtrlType;        /* Interrupt controller type */
   unsigned char  uchReserved[3];     /* Reserved (padded with 0)  */
   unsigned char  IntMask[4];         /* PCI INT mask              */
                                      /*   For 8259:               */
                                      /*       Bit 0 : INTA        */
                                      /*       Bit 1 : INTB        */
                                      /*       Bit 2 : INTC        */
                                      /*       Bit 3 : INTD        */
                                      /*       Others: Reserved    */
   unsigned char  IRQ[4];             /* corresponding IRQ number  */
                                      /*   to the IntMask bits set */
   } IntStruct;

typedef struct _IntMap {              /* PCI Int to 8259 or MPIC IRQ conversion map */
   unsigned char  SlotNumber;         /* Slot number engraved on the box */
                                      /*   zero indicates invalid entry   */
   unsigned char  DevFuncNumber;      /* PCI slot's DeviceFunction number */
                                      /*   zero indicates no option slot  */
                                      /*   non-zero indicates a valid option slot */
   unsigned char  uchReserved[2];     /* Reserved (padded with 0)    */
   IntStruct      isInt[MAX_PCI_INTS]; /* Interrupt mapping table     */
                                      /*   0, 0 indicates last valid */
                                      /*   entry. Entries after 0, 0 */
                                      /*   must be ignored           */
   } IntMap;

typedef struct _PCIInfoPack {
   unsigned char  Tag;                /* large tag = 0x84 Vendor specific  */
   unsigned char  Count0;             /* lo byte of count                  */
   unsigned char  Count1;             /* hi byte of count                  */
      /* count = number of pluggable PIC slots * sizeof(IntMap) + 6        */
   unsigned char  Type;               /* = 3 (PCI bridge)                  */
   unsigned char  ConfigBaseAddress[4];/* Base address of PCI Configuration */
   unsigned char  BusNumber;          /* PCI Bus number                    */
```

```
    IntMap         Map[1];             /* Interrupt map array for each PCI  */
                                       /*  slots that are pluggable         */
    } PCIInfoPack;

#endif  /* ndef _PCIPNP_ */
```

The structure below is the Plug and Play definition for L2 cache devices.

```
/* Structure map for L2 cache in PnP Vendor specific packet */

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

#ifndef _L2PNP_
#define _L2PNP_

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

typedef enum _L2_Store_Algorithm {
   None = 0,
   WriteThru = 1,
   CopyBack = 2,
   } L2_Store_Algorithm;

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

typedef enum _L2_HW_Assist {
   Invalidate = 1,                     /* invalidate        */
   Flush = 2,                          /* flush             */
   L2PowerManaged = 4                  /* power managed     */
   } L2_HW_Assist;

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

#endif  /* ndef _L2PNP_ */
```

The structure below is the Plug and Play definition for processor chip identities.

```
/* Structure map for Chip ID in PnP Vendor specific packet */

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

#ifndef _CHPIDPNP_
#define _CHPIDPNP_

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

typedef enum _Chip_ID {

   /* PCI Bridge chips */
   PCI_Br1 = 1,                         /* Part # IBM27-82650-653/4        */
   PCI_Br2 = 2,                         /* Part # IBM27-82657-50           */
   PCI_Br3 = 3,                         /* Part # MPC105                   */

   /* Power management chips (ISA) */
   PM_ISA_1 = 1,                       /*  Part # ---                      */
   Sig750 = 2,                         /*     Signetic 87C750             */

   /* Power management chips (PCI) */
   PM_PCI_1 = 1,                        /* Part # ---                     */

   /* L2 Cache controller */
   L2_Cntl_1 = 1,                      /*  Part # IBM27-82681-66        */
   L2_Cntl_2 = 2,                      /*  Part # for Energy managed WS */
   L2_Cntl_3 = 3,                      /*  Part # for Portable WS       */
   L2_Cntl_4 = 4,
   L2_Cntl_5 = 5,
   L2_Cntl_6 = 6,
   L2_Cntl_7 = 7

   } Chip_ID;

typedef struct _ChipIDPack {           /* This packet identifies chip set ID. */
                                       /* Details of these specifics are      */
                                       /* documented in the chip specs.       */
   unsigned char  Tag;                 /* small tag = 0x7n with n bytes       */
   unsigned char  Type;                /*    = 1 (Chip ID)                    */
   unsigned char  VendorID0;           /*    Bit(7)=0                         */
                                       /*    Bits(6:2)=1st character in       */
                                       /*       compressed ASCII              */
                                       /*    Bits(1:0)=2nd character in       */
                                       /*       compressed ASCII bits(4:3)    */
   unsigned char  VendorID1;           /*    Bits(7:5)=2nd character in       */
                                       /*       compressed ASCII bits(2:0)    */
                                       /*    Bits(4:0)=3rd character in       */
                                       /*       compressed ASCII              */
   unsigned char  Name[2];             /*    Chip ID name                     */
   } ChipIDPack;

#endif  /* ndef _CHPIDPNP_ */
```

The structure below is the Plug and play definition for Disk drives.

```
/* Structure map for Diskette drives Vendor specific packet */

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

#ifndef _DSKTPNP_
#define _DSKTPNP_

#define  Dskt_Packet 0x84               /* tag for DsktInfoPack          */

typedef struct _DsktInfoPack {
   unsigned char  Tag;                  /* large tag = 0x84 Vendor specific */
   unsigned char  Count0;               /* lo byte of number of drives + 1  */
   unsigned char  Count1;               /* hi byte of number of drives + 1  */
            /* Count (Count0 and Count 1) - 1 = number of diskette drives    */
   unsigned char  Type;                 /*   = 1 (diskette)                 */
   unsigned char  Dskt[1];              /* diskette drives info array       */
                                        /*      0 : no drive present        */
                                        /*      1 : 3.5" 2MB drive          */
                                        /*      2 : 3.5" 4MB drive          */
                                        /*      3 : 5.25" 1.6MB drive       */
   } DsktInfoPack;

#endif  /* ndef _DSKTPNP_ */
```

Appendix K. Dump of Residual Data

A dump of the residual data constructed by a machine which matches the Reference Implementation is shown below:

```
Residual ID:
   Length = 0x6a0c
   Version = 0
   Revision = 0

Residual VPD:
   Model  = IBM PPS Model 6015
   Serial = ffffffffffffffff
   Spec Version  = 0
   Spec Revision = 0
   FW Support    = 0x1c5
   NVRAM size   = 4096
   SIMM slots   = 6
   ISA slots    = 3
   PCI slots    = 2
   PCMCIA slots = 0
   MCA slots    = 0
   EISA slots   = 0
   CPU MHz      = 66
   CPU bus MHz  = 66
   PCI bus MHz  = 33
   Time base div = 0xffffffff
   Word width = 32
   Page size  = 4096
   Block size = 32
   Granule size = 32
   Cache size   = 32
   Cache attrib = 2
   Cache assoc  = 8
   Cache line   = 64
   I-Cache size   = 32
   I-Cache assoc  = 8
   I-Cache line   = 64
   D-Cache size   = 32
   D-Cache assoc  = 8
   D-Cache line   = 64
   TLB size   = 256
   TLB attrib = 2
   TLB assoc  = 2
   I_TLB size   = 256
   I_TLB assoc  = 2
   D_TLB size   = 256
   D_TLB assoc  = 2
   ExtVPD ptr = 0x0

Residual CPU:
   Num CPUs = 1
     CPU[0] CPU Type   = 0x   10002
            CPU Serial = 0xffffffff
            L2 size    = 0
            L2 assoc   = 0
```

```
| Residual Memory:
|    Total Memory = 0x2000000
|    Good Memory  = 0x2000000
|    Mem seg[0] =
|      Mem seg is Firm Code
|      Mem seg base page  = 0x0
|      Mem seg page count = 0x400
|    Mem seg[1] =
|      Mem seg is Firm Heap
|      Mem seg base page  = 0x400000
|      Mem seg page count = 0x80
|    Mem seg[2] =
|      Mem seg is Free
|      Mem seg base page  = 0x480000
|      Mem seg page count = 0x1b80
|    Mem seg[3] =
|      Mem seg is Unpopulated
|      Mem seg base page  = 0x2000000
|      Mem seg page count = 0x7e000
|    Mem seg[4] =
|      Mem seg is SystemIO
|      Mem seg is ISAAddr
|      Mem seg base page  = 0x80000000
|      Mem seg page count = 0x800
|    Mem seg[5] =
|      Mem seg is SystemIO
|      Mem seg is PCIConfig
|      Mem seg base page  = 0x80800000
|      Mem seg page count = 0x800
|    Mem seg[6] =
|      Mem seg is SystemIO
|      Mem seg is PCIAddr
|      Mem seg base page  = 0x81000000
|      Mem seg page count = 0x3e800
|    Mem seg[7] =
|      Mem seg is SystemIO
|      Mem seg is SystemRegs
|      Mem seg base page  = 0xbf800000
|      Mem seg page count = 0x800
|    Mem seg[8] =
|      Mem seg is IOMemory
|      Mem seg base page  = 0xc0000000
|      Mem seg page count = 0x3f000
|    Mem seg[9] =
|      Mem seg is IOMemory
|      Mem seg base page  = 0xff000000
|      Mem seg page count = 0x800
|    Mem seg[10] =
|      Mem seg is UnPopSystemROM
|      Mem seg base page  = 0xff800000
|      Mem seg page count = 0x700
|    Mem seg[11] =
|      Mem seg is SystemROM
|      Mem seg base page  = 0xfff00000
|      Mem seg page count = 0x80
|    Mem seg[12] =
|      Mem seg is UnPopSystemROM
|      Mem seg base page  = 0xfff80000
|      Mem seg page count = 0x80
```

```
|     Mem seg[13] =
|       Mem seg is Boot Image
|       Mem seg base page  = 0x3b57e0
|       Mem seg page count = 0x4b
|     Total SIMMs = 4
|       SIMM[0] size = 8
|       SIMM[1] size = 8
|       SIMM[2] size = 8
|       SIMM[3] size = 8
|       SIMM[4] size = 0
|       SIMM[5] size = 0

|  Residual Devices:
|     Total Devices = 17
|     Device [0] = [PNPB00F] Crystal CS4231 Audio Device
|       Dev ID = 0x fb0d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 4
|       Sub   Type = 1
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 0
|       Possible   = 16
|       Compatible = 17
|     Device [1] = [PNP0700] PC Standard Floppy Disk Controller
|       Dev ID = 0x   7d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 1
|       Sub   Type = 2
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 18
|       Possible   = 29
|       Compatible = 30
|     Device [2] = [PNP0200] AT DMA Controller
|       Dev ID = 0x   2d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 8
|       Sub   Type = 1
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 31
|       Possible   = 152
|       Compatible = 153
```

```
|     Device [3] = [PNP0000] AT Interrupt Controller
|       Dev ID = 0x    d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 8
|       Sub   Type = 0
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 154
|       Possible   = 166
|       Compatible = 167
|     Device [4] = [PNP0A00] ISA Bus
|       Dev ID = 0x   ad041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 6
|       Sub   Type = 1
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 168
|       Possible   = 169
|       Compatible = 179
|     Device [5] = [PNP0400] Standard LPT Parallel Port
|       Dev ID = 0x   4d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Configurable
|       Device is Disableable
|       Device is Input
|       Device is Output
|       Base  Type = 7
|       Sub   Type = 1
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 180
|       Possible   = 188
|       Compatible = 214
|     Device [6] = [PNP0A03] PCI Bus
|       Dev ID = 0x 30ad041
|       Serial = 0xffffffff
|       Device is PCI
|       Device is Static
|       Base  Type = 6
|       Sub   Type = 4
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|
```

```
|       Bus Access[2] = 0
|       Allocated  = 215
|       Possible   = 219
|       Compatible = 220
|     Device [7] = [PNP0B00] AT RTC
|       Dev ID = 0x   bd041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 8
|       Sub   Type = 3
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 221
|       Possible   = 229
|       Compatible = 230
|     Device [8] = [PNPA00F] NCR 810 SCSI Controller
|       Dev ID = 0x fa0d041
|       Serial = 0xffffffff
|       Device is PCI
|       Device is Configurable
|       Device is Disableable
|       Base  Type = 1
|       Sub   Type = 0
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 128
|       Bus Access[1] = 16
|       Bus Access[2] = 0
|       Allocated  = 231
|       Possible   = 235
|       Compatible = 236
|     Device [9] = [PNP0501] 16550A Compatible Serial port
|       Dev ID = 0x 105d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Configurable
|       Device is Disableable
|       Device is Input
|       Device is Output
|       Base  Type = 7
|       Sub   Type = 0
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 237
|       Possible   = 245
|       Compatible = 319
|     Device [10] = [PNP0501] 16550A Compatible Serial port
|       Dev ID = 0x 105d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Configurable
|       Device is Disableable
```

```
|       Device is Input
|       Device is Output
|       Base  Type = 7
|       Sub   Type = 0
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 320
|       Possible   = 328
|       Compatible = 402
|     Device [11] = [PNP0100] AT Timer
|       Dev ID = 0x   1d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Base  Type = 8
|       Sub   Type = 2
|       Inter Type = 1
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 403
|       Possible   = 415
|       Compatible = 416
|     Device [12] = [PNP0303] IBM Enhanced (101/102 key, PS/2 mouse)
|       Dev ID = 0x 303d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Device is Disableable
|       Device is Removable
|       Device is ConsoleIn
|       Device is Input
|       Base  Type = 9
|       Sub   Type = 0
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 417
|       Possible   = 429
|       Compatible = 430
|     Device [13] = [PNP0F03] Microsoft PS/2 Mouse
|       Dev ID = 0x 30fd041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Static
|       Device is Disableable
|       Device is Removable
|       Device is Input
|       Base  Type = 9
|       Sub   Type = 2
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
```

```
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 431
|       Possible   = 443
|       Compatible = 444
|     Device [14] = [PNP090E] Weitek P9000 Graphics Adapter
|       Dev ID = 0x e09d041
|       Serial = 0xffffffff
|       Device is PCI
|       Device is Configurable
|       Device is Disableable
|       Device is ConsoleOut
|       Base  Type = 3
|       Sub   Type = 0
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 6
|       Bus Access[2] = 0
|       Allocated  = 445
|       Possible   = 449
|       Compatible = 450
|     Device [15] = [PNP8327] IBM Token Ring (All types)
|       Dev ID = 0x2783d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Input
|       Device is Output
|       Base  Type = 2
|       Sub   Type = 1
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 451
|       Possible   = 480
|       Compatible = 481
|     Device [16] = [PNP8327] IBM Token Ring (All types)
|       Dev ID = 0x2783d041
|       Serial = 0xffffffff
|       Device is ISA
|       Device is Input
|       Device is Output
|       Base  Type = 2
|       Sub   Type = 1
|       Inter Type = 0
|       Spare Type = 0
|       Bus Access[0] = 0
|       Bus Access[1] = 0
|       Bus Access[2] = 0
|       Allocated  = 482
|       Possible   = 511
|       Compatible = 512

|  Residual Device Heap:

|    22   0   4  4b   8  30   4  2a  40  52
|    2a  80  52  71   1  78  78  78  22  40
```

```
|     0  4b   3  f0   8  2a   4  48  78  78
|    78  4b   0   0  10  4b   0  80  11  4b
|     0  94   3  4b   0  98   1  4b   0  9c
|     4  4b   0  c0   1  4b   0  c2   1  4b
|     0  c4   1  4b   0  c6   1  4b   0  c8
|     1  4b   0  ca   1  4b   0  cc   1  4b
|     0  ce   1  4b   0  d0   1  4b   0  d2
|     1  4b   0  d4   1  4b   0  d6   1  4b
|     0  d8   1  4b   0  da   1  4b   0  dc
|     1  4b   0  de   1  4b   4   b   1  4b
|     4  10   4  4b   4  15   7  4b   4  1d
|    13  4b   4  34   c  4b   4  81   3  4b
|     4  87   1  4b   4  89   3  4b   4  d6
|     1  78  78  78  22   4   0  4b   0  20
|     2  4b   0  a0   2  78  78  78  78  22
|     0   2  22   0   8  22   0  40  78  78
|    22  80   0  4b   3  bc   3  78  30  22
|    80   0  4b   3  bc   3  30  22  80   0
|    4b   3  78   6  30  22  20   0  4b   2
|    78   6  38  78  78  22   0  80  78  78
|    78  22   0   1  4b   0  70   2  78  78
|    78  22   0  20  78  78  78  22  10   0
|    4b   3  f8   8  78  30  22  10   0  4b
|     3  f8   8  30  22   8   0  4b   2  f8
|     8  30  22  10   0  4b   2  20   8  30
|    22  10   0  4b   2  e8   8  30  22  10
|     0  4b   3  38   8  30  22  10   0  4b
|     3  e8   8  30  22  10   0  4b   2  38
|     8  30  22  10   0  4b   2  e0   8  30
|    22  10   0  4b   2  28   8  38  78  78
|    22   8   0  4b   2  f8   8  78  30  22
|    10   0  4b   3  f8   8  30  22   8   0
|    4b   2  f8   8  30  22  10   0  4b   2
|    20   8  30  22  10   0  4b   2  e8   8
|    30  22  10   0  4b   3  38   8  30  22
|    10   0  4b   3  e8   8  30  22  10   0
|    4b   2  38   8  30  22  10   0  4b   2
|    e0   8  30  22  10   0  4b   2  28   8
|    38  78  78  22   1   0  4b   0  40   4
|    4b   0  78   4  78  78  78  22   2   0
|    4b   0  60   1  4b   0  64   1  78  78
|    78  22   0  10  4b   0  60   1  4b   0
|    64   1  78  78  78  22   0  80  78  78
|    78  4b   a  20   0  81   9   0  21   0
|     0   0   0   0  10   0   0  81   9   0
|     1   0   0   0   0   0  10   0   0  78
|    78  78  4b   0   0   0  81   9   0  21
|     0   0   0   0   0  10   0   0  81   9
|     0   1   0   0   0   0   0  10   0   0
|    78  78  78   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
|     0   0   0   0   0   0   0   0   0   0
```

Obtaining Additional Information

Several sources exist for obtaining additional information about the PowerPC microprocessor and the PowerPC Reference Platform.

Anyone interested in obtaining more information on the PowerPC processor or on the PowerPC Reference Platform may use the following sources:

- IBM at 1-800-PowerPC (1-800-769-3772) in the U.S.A (MPR-PPC-RPU-02 is the number for the *PowerPC Reference Platform Specification*, Version 1.0)
- If the 1-800-PowerPC number can not be reached or if multilingual operators are required, use 1-708-296-6767
- IBM at (39)-39-600-4295 in Europe
- Motorola at 1-800-845-MOTO (6686)

Copies of the *PowerPC Reference Platform Specification* may be obtained from these numbers. Hardware system vendors may obtain information on IBM components or the IBM design kits which give further information on the reference implementation by contacting IBM at the numbers listed above or at the following numbers:

- Within Europe (33)-6713-5757 in French
- Within Europe (33)-6713-5756 in Italian
- Within Europe (49)-511-516-3444 in English
- Within Europe (49)-511-516-3555 in German
- In Asia (81)-755-87-4745 in Japanese

An electronic forum on CompuServe has been established for the discussion of PowerPC Reference Platform topics and for obtaining answers to questions on the PowerPC Reference Platform. Go to the "PowerPC" forum on CompuServe and join the "Reference Platform" topic. The *PowerPC Reference Platform Specification* in PostScript format is maintained on a library of the PowerPC forum.

Several white papers are available from IBM at 1-512-838-5552. These papers expand on the information in this specification. The papers which were available at the time of publishing this specification are as follows:

- *PowerPC 60x/PCI Bus Bridge Implementation for PowerPC Reference Platform* by Yongjae Rim
- *Bi-Endian Designs in PowerPC Reference Platform* by Shien-Tai Pan
- *Symmetric Multiprocessing* by Don McCauley
- *PowerPC Endian Switch Code* by Gary Tsao
- *Plug and Play for PowerPC Reference Platform* by Gary Tsao
- *L2 Cache Design for PowerPC* by Allan Steel

This specification, the white papers, and various data structures (e.g. NVRAM Header) are available via anonymous FTP. The server address is ftp.austin.ibm.com and the material is placed in the directory /pub/technology/spec.

A paper, *The PowerPC Reference Platform Specification and Machine Abstraction* by Allan Steel, is available in the Spring Compcon 94 digest.

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

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

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

The PowerPC Architecture, ISBN 1-55860-316-6, is published by Morgan Kaufmann Publishers, 415-392-2665. It is available in some bookstores and may be ordered from local IBM publications sources at 1-800-426-6477 in the US only.

Kernel Extensions and Device Support Programming Concepts, IBM document number SC232207, and *Writing a Device Driver for AIX Version 3.2*, IBM document number GG243629, may be ordered by calling 1-800-879-2755.

/ To purchase a copy of *System V Application Binary Interface*, call 1-800-947-7700 in the U.S.A. or 515-284-6751
/ outside the U.S.A.

/ To order the *Guide to Mac Family Hardware*, call 1-800-282-2732.

/ Back issues of the *Microsoft Systems Journal* may be ordered by calling 1-800-444-4881 in the U.S.A. or 415-715-6213
/ outside the U.S.A.

/ To purchase the *MS-DOS Programmer's Reference*, call 1-800-677-7377.

| Call 212-642-4900 for orders or inquiries pertaining to ANSI and ISO standards.

To order copies of EIA standards, contact Global Engineering Documents at 1-800-854-7179.

Copies of IEEE standards may be obtained by calling 1-800-678-IEEE. For information about IEEE standards, call 908-562-3800.

/ To purchase PCI documents, call 1-800-433-5177 in the U.S.A. or 503-797-4207 outside the U.S.A.

Copies of PCMCIA standards may be obtained by calling 408-720-0107.

The PowerOpen Association may be contacted at the address and telephone number listed below:

PowerOpen Association
10500 North Wolfe Road
Suite SW2-255
Cupertino, CA 95014
1-800-457-0463

Bibliography

Documents which were referenced in this specification are listed below:

- *The PowerPC Architecture*, ISBN 1-55860-316-6
- *Kernel Extensions and Device Support Programming Concepts*, IBM order number SC232207
- *Writing a Device Driver for AIX Version 3.2*, IBM order number GG243629
- *How To Create Endian-Neutral Software for Portability*, TR54.837 (also published in *Dr. Dobb's Journal* for October and November 1994)
- PowerPC processor-specific user's manuals:
  - *PowerPC 601 RISC Microprocessor User's Manual*, IBM order number 52G7484, Motorola order number MPC601UM/ADREV1
- *ANSI Standard X3.131-1990 (Revision 10c) for SCSI-2*
- *ANSI Standard X3.221 AT Attachment (Revision 4a)*
- *ANSI/NISO/ISO 9660, Information Processing -- Volume and File Structure of CD-ROM for Information Interchange*
- EIA/TIA-232-E, *Interface Between Data Terminal Equipment and Data Circuit Terminated Equipment Employing Serial Binary Data Interchange*
- EIA-422-A, *Electrical Characteristics of Balanced Voltage Digital Interface Circuits*
- IEEE P1275, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices*
- IEEE P1284, *Standard Signaling Method for a Bi-Directional Parallel Peripheral Interface for Personal Computers*, March 15, 1993
- IEEE 802.3, ISO/IEC 8802-3, *Information technology -- Local and metropolitan area networks Part III: Carrier sense multiple access with collision detection (CSMA-CD) access method and physical layer specifications*, 1993
- IEEE 802.5, *Standards for Local Area Networks: Token Ring Access Method and Physical Layer Specifications*
- IEEE 996, *A Standard for an Extended Personal Computer Back Plane Bus*
- *ISO 639, Code for the representation of names of languages*
- *Microsoft Hardware Abstraction Layer*, Beta Version, March 1993
- PCI documents (available from and maintained by the PCI Special Interest Group):
  - *PCI Local Bus Specification*, Revision 2.0, April 30, 1993
  - *PCI System Design Guide*, Revision 1.0, September 1993
  - *PCI to PCI Bridge Architecture Specification*, Revision 1.0, April 5, 1994
  - *PCI Multimedia Design Guide*, Revision 1.0, March 29, 1994
- PCMCIA standards (can be ordered from the Personal Computer Memory Card International Association):
  - *PC Card Standard Specification*, Release 2.1, July 1993
  - *Socket Services Specification*, Release 2.1, July 1993
  - *Card Services Specification*, Release 2.1, July 1993
  - *PC Card ATA Mass Storage Specification*, Release 1.02, July 1993
  - *AIMS Specification*, Release 1.01, November 1992
  - *Recommended Extensions*, Release 1.00, November 1992
- Michel Dubois, Christoph Scheurich, and Faye Briggs, *Memory Access Buffering in Multiprocessors*, Proceedings of the 13th ISCA, pp. 434-441, 1986.
- *PowerOpen ABI*
- *PowerOpen API*
- *System V Application Binary Interface*
- *MS-DOS Programmer's Reference*

- “Peering Inside the PE: A Tour of the Win32 Portable Executable File Format,” *Microsoft Systems Journal*, March 1994
- *Bootstrap Protocol*, Internet RFC 951.

Acronyms and Abbreviations

Table 40 (Page 1 of 4). Acronyms and Abbreviations

Term	Definition
ABI	Application binary interface
ANSI	American National Standards Institute
API	Applications programming interface
APM	Available processor mask
ASIC	Application-specific integrated circuit
ATM	Asynchronous transfer mode
BAT	Block address translation
BE	Big-Endian
BEPI	Block effective page index
BL	Block length
BLR	Branch to link register
BRPN	Block real page number
BTAS	Boot-time abstraction software
BUID	Bus unit identifier
CB	Copy-back
CISC	Complex instruction set computer
CPPR	Current processor priority register
CRC	Cyclic redundancy check
CV	Compatible value
DAC	Digital-to-analog converter
DBAT	Data block address translation register
DCE	Distributed computing environment
DDI	Device driver interface
DDK	Device driver development kit
DMA	Direct memory access
DRAM	Dynamic random access memory
DSI	Data storage interrupt
DSP	Digital signal processor
EA	Effective address
ECC	Error checking and correcting
ECP	Extended capabilities port
EEPROM	Electrically erasable programmable read-only memory
EISA	Extended industry standard architecture
ELF	Executable and linking format
EOI	End of interrupt
EPLD	Electrically programmable logic device
EPROM	Erasable programmable read-only memory

Table 40 (Page 2 of 4). Acronyms and Abbreviations	
Term	**Definition**
FAL	Firmware abstraction layer
FAT	File allocation table
FCS	Fiber Channel Standard
FDDI	Fiber distributed data interface
FIFO	First in first out
FRU	Field-replaceable unit
GB	Gigabyte
GMT	Greenwich mean time
GOT	Global offset table
GPR	General purpose register
GUI	Graphical user interface
HAL	Hardware abstraction layer
HAS	Hardware abstraction software
HPFS	High-performance file system
HTAB	Hashed page table
IBAT	Instruction block address translation register
IC	Integrated circuit
IDE	Integrated device electronics
IP	Interrupt prefix
IPIRR	Inter-processor interrupt request register
IRDT	Interrupt redirection table
IRQL	Interrupt request level
IRQP	Interrupt request priority
IRR	Interrupt request register
ISA	Industry standard architecture
ISBN	International standard book number
ISDN	Integrated services digital network
ISE	Instruction set emulator
ISI	Instruction storage interrupt
ISR	Interrupt source register
ITL	Independent test lab
JEIDA	Japan Electronic Industry Development Association
JFS	Journaled file system
KB	Kilobyte
KBI	Kernel binary interface
KPI	Kernel programming interface
L1	First-level cache
L2	Second-level cache
LAN	Local area network
LBX	Local branch exchange

Table 40 (Page 3 of 4). Acronyms and Abbreviations

Term	Definition
LE	Little-Endian
LSb	Least significant bit
LSB	Least significant byte
LVM	Logical volume manager
MB	Megabyte
MCA	Micro Channel Architecture
MIDI	Musical instrument digital interface
MK	Microkernel
MMU	Memory management unit
MP	Multiprocessor
MSb	Most significant bit
MSB	Most significant byte
MSR	Machine status register
NFS	Network file system
NVRAM	Non-volatile random access memory
OEM	Original equipment manufacturer
ONC	Open network computing
OS	Operating system
PCI	Peripheral component interconnect
PCIB/MC	PCI bridge and memory controller
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal digital assistant
PE	Portable executable
PIM	Platform-independent module
PLL	Phase lock loop
PnP	Plug and Play
POE	PowerOpen Environment
POST	Power-on self test
PSM	Platform-specific module
PTE	Page table entry
QFP	Quad flat pack
QIC	Quarter-inch cartridge
RA	Real address
RAID	Redundant array of independent disks
RAMDAC	Random access memory and digital-to-analog converter
RBA	Relative block address
RFC	Request for comments
RGB	Red-green-blue
RISC	Reduced instruction set computer
RPN	Real page number

Table 40 (Page 4 of 4). Acronyms and Abbreviations	
Term	**Definition**
RTAS	Run-time abstraction software
SCSI	Small computer system interface
SIMM	Single inline memory module
SIO	Standard I/O
SMP	Symmetric multiprocessor
SODIMM	Small outline dual inline memory module
SPARC	Scalable processor architecture
SPR	Special purpose register
SR	Segment register
SSD	Storage system division
SVID	System V interface definition
TCP/IP	Transmission control protocol internet protocol
TEA	Transaction error acknowlegement
TLB	Translation lookaside buffer
TTY	Teletypewriter
UMCU	Universal micro control unit
VA	Virtual address
VL	Network VESA local bus
VME	VERSA Module Eurocard
VPD	Vital product data
VPN	Virtual page number
VSID	Virtual segment ID
WT	Write-through

Glossary

Alpha-numeric input device. A device for the input of symbols and alphabetical and numeric characters, such as a keyboard.

BAT register. A mechanism which provides a means of mapping ranges of virtual addresses larger than a page onto contiguous areas of real storage.

Big-Endian. A byte ordering method in which the most significant byte is stored first.

Bi-Endian. Having Big-Endian and Little-Endian byte ordering capability.

Dataless. A hardware configuration in which the system's hardfile is used only for system support; most of the system software and data reside on a network-connected storage device.

Directly attached. Electrically connected.

Hard disk. A storage media consisting of several spinning platters on which information is read and written electronically.

Hardfile. A storage media for reading and writing large volumes of data. Typically implemented as a hard disk, but other storage technologies may be used.

HUMAN-CENTERED. Technology which is centered around the human senses of sight, sound and touch.

Hot-plug capability. The ability to couple a peripheral device to a system without shutting down or restarting the system.

Little-Endian. A byte ordering method in which the least significant byte is stored first.

Medialess. A hardware configuration with no data storage capability. A network connection supplies storage for the operating system, applications and data.

Microkernel. A small, message-passing nucleus of system software running in the most privileged state of the computer.

Pointing device. A device which provides two-dimensional positioning, such as a mouse, tablet or touch screen.

Trademark Information

The following terms, denoted by a double asterisk (**) in this publication, are trademarks or registered trademarks of the companies shown:

/	DeskSet	Sun Microsystems, Inc.
	EtherCard	Standard MicroSystems Corporation
	EtherExpress	Intel Corporation
	EtherLink	3Com Corporation
	Ethernet	Xerox
	LocalTalk	Apple Computer, Inc.
	Macintosh	Apple Computer, Inc.
	NetWare	Novell, Inc.
/	Newscard	Motorola, Inc.
/	NFS	Sun Microsystems, Inc.
/	ONC	Sun Microsystems, Inc.
/	OpenWindows	Sun Microsystems, Inc.
	PostScript	Adobe Systems Incorporated
	SatisFAXtion	Intel Corporation
/	ScanJet	Hewlett-Packard
	Solaris	Sun Microsystems, Inc.
	Sound Blaster	Creative Technology, Ltd.
/	SunOS	Sun Microsystems, Inc.
/	ToolTalk	Sun Microsystems, Inc.
|	UNIX	X/Open Company, Ltd.
/	Wabi	Sun Microsystems, Inc.
	WangDAT	WangDAT Inc.
	Windows	Microsoft Corporation
	Windows NT	Microsoft Corporation
	X Window System	Massachusetts Institute of Technology

Index

I

K

L

M

N

O

P

R

S

T

U

V

W

END OF DOCUMENT

This is the last page of this document