Contents—ABIOS

© Copyright IBM Corp. 1991

Section 1. Introduction to Advanced BIOS

© Copyright IBM Corp. 1991

Notes:

Introduction

Advanced BIOS (ABIOS) is firmware that isolates an operating system from the low-level system hardware interface in IBM Personal System/2 systems that use 80286 or 80386 microprocessors. The operating system makes functional requests of ABIOS (read or write) instead of directly manipulating the I/O ports and control words of the system hardware. This enables details of the hardware attachments and the timings of the hardware interfaces to be altered without disturbing the operating-system components above the ABIOS interface.

ROM BIOS operates as a single-tasking component in which addressing capabilities are limited to less than 1 megabyte of memory and only in the real-address mode (real mode) of the Intel microprocessor. ABIOS supports addressing above 1 megabyte, using the protected virtual address mode (protected mode) of the Intel microprocessor. ABIOS is contained in ROM but does not prevent a RAM implementation. ABIOS can be operated in the real mode, in the protected mode, or in a bimodal environment using both the real mode and the protected mode. ABIOS provides a data structure for implementing a protected-mode or bimodal (real and protected modes) operating system. In addition, ABIOS can run in virtual 8086 mode.

Requests to ABIOS that are made by an operating system fall into three categories: single-staged, discrete multistaged, and continuous multistaged. Single-staged requests perform the requested function before returning to the caller. Discrete multistaged requests start an action or operation that involves a delay before the operation is completed. Continuous multistaged requests start an action or operation that also involves a delay but never ends. For multistaged operations, control is returned to the caller during these delays so that the processing time can be used. An interrupt from an I/O device usually indicates completion of a stage of the operation.

The following figure shows the three categories of ABIOS requests.

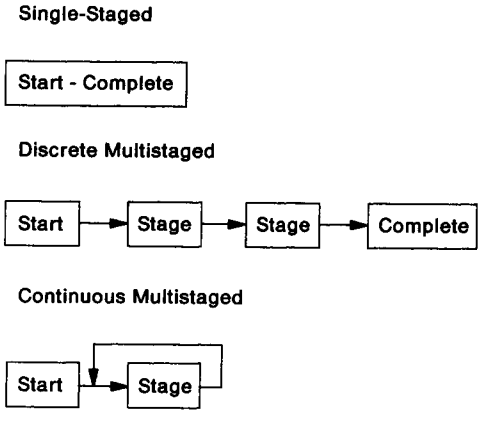


Figure 1-1. Types of Requests

Data Structures

Requests to ABIOS that are made by an operating system are made through transfer conventions that are provided by the ABIOS structure. Transfer conventions require data structures that link the operating system to the device-function routines of each supported device. These data structures are the common data area, function transfer tables, and device blocks. They reside in system memory and are initialized during ABIOS initialization.

Transfer conventions that are provided by ABIOS are defined to enable operations that use the real mode, the protected mode, or both modes of an Intel microprocessor. To provide flexibility in implementing a real-mode, protected-mode, or bimodal operating system, the common data area links all ABIOS pointers into a single structure (see "Common Data Area" on page 2-3). This structure contains the function-transfer-table pointers, the device-block pointers, and the ABIOS data pointers.

ABIOS entry points are stored in vector tables called function transfer tables (see "Function Transfer Table" on page 2-7). Each supported ABIOS device has an associated function transfer table. The first three entries of the function transfer table are structured entry routines: the Start routine, the Interrupt routine, and the Time-Out routine.

ABIOS routines require a permanent work area, called a device block, for each device (see "Device Block" on page 2-9). Hardware port addresses, interrupt levels, and device-status information are stored in a device block.

Initialization

Initialization is a defined protocol between ABIOS and an operating system. The operating system plays a major role in the initialization process, including starting the process. Until the operating system starts the initialization process, ABIOS cannot be used (see Section 3, "Initialization"). The initialization process must occur in the real mode of the microprocessor. It consists of three steps:

1. The operating system calls BIOS to build the system parameters table. This table describes the number of devices that are available in the system, common entry points, and system-stack requirements.
2. The operating system calls BIOS to build the initialization table. This table defines the initialization information for each device that the system supports. This information is used to initialize device blocks and function transfer tables.
3. The operating system allocates memory for the common data area, using the initialization information that was returned in step 2. The memory for device blocks and function transfer tables is allocated, and the device-block pointers and function-transfer-table pointers are initialized in the common data area. Then the operating system calls ABIOS to build a device block and function transfer table for each device.

The flow of the initialization process is shown below.

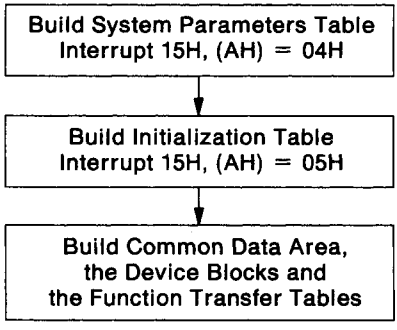


Figure 1-2. Flow of the Initialization Process

Transfer Conventions

After ABIOS is initialized, requests are presented through a parameter block, called a request block. A request block has fields that identify the target device, the requested operation, details of the request, memory locations that are involved in a data transfer, and the status of the staged or completed request. Request blocks are described in detail in Section 4, "Transfer Conventions."

ABIOS is implemented as a call-return programming model, using either the ABIOS transfer convention or the operating-system transfer convention. These two calling conventions give an operating system flexibility in calling ABIOS. Both calling conventions use stacks to pass request information to the target ABIOS device routine.

The ABIOS transfer convention is the simplest calling sequence for the operating system. The operating system passes the common-data-area pointer and the request-block pointer to one of three common entry points: the Common Start routine, the Common Interrupt routine, and the Common Time-Out routine. The pointers to these common routines are returned to the operating system during initialization. The common routines use the request-block information and the common-data-area pointer to get the device-block pointer and the function-transfer-table pointer from the common data area. The common routine then transfers control to the requested ABIOS routine whose pointer is in the function transfer table.

The flow of the ABIOS transfer convention is shown below.

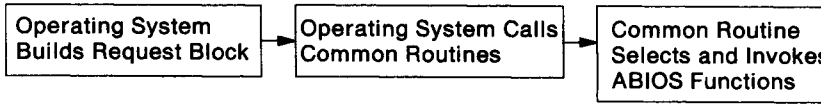


Figure 1-3. Flow of ABIOS Transfer Convention

The operating-system transfer convention requires the operating system to determine the address for the requested ABIOS routine. This gives the operating system flexibility in maintaining ABIOS-routine addresses that are frequently called. This method is useful for handling interrupts from character and programmed-I/O devices that repeatedly call a single routine. The common-data-area pointer, the request-block pointer, the function-transfer-table pointer, and the device-block pointer are required on entry to the ABIOS routine.

The flow of the operating-system transfer convention is shown below.

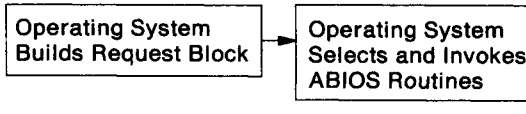


Figure 1-4. Flow of Operating-System Transfer Convention

The ABIOS transfer convention and the operating-system transfer convention are described in detail in Section 4, "Transfer Conventions."

Interrupt Processing

For multistaged requests, interrupts from hardware devices cause the microprocessor to branch to predefined addresses in the interrupt vector table. When an interrupt occurs, ABIOS expects the operating system to receive control. ABIOS provides interrupt routines for the processing of ABIOS interrupts. Interrupt processing is described in "Interrupt Processing" on page 5-3.

Extending ABIOS

The ability to add, patch, extend, and replace ABIOS routines is necessary for supporting new devices or device features on the system. For more information, see Section 5, "Additional Information."

Section 2. Data Structures

© Copyright IBM Corp. 1991

Notes:

Introduction

ABIOS uses data structures to link the operating system to the device-function routines for each ABIOS device. These data structures are the common data area, the function transfer table, and the device block. They reside in system memory and are initialized during ABIOS initialization.

Transfer conventions that are provided by ABIOS are defined to enable operations that use the real mode, the protected mode, or both modes of an Intel microprocessor. ABIOS provides the common data area for implementing a real-mode, protected-mode, or bimodal operating system. This structure contains the function-transfer-table pointers, the device-block pointers, and the ABIOS data pointers. The common data area links all ABIOS pointers in a single structure to enable an operating system to manage ABIOS requests in both operating environments of the Intel microprocessors.

Common Data Area

The common data area contains data pointers that facilitate the ABIOS operation in a bimodal environment. These data pointers are established during ABIOS initialization and contain information for each device that the system supports. The common data area is required in all three operating modes.

Each ABIOS device has a physical-device identifier, called a device ID. A device ID has one or more logical IDs that serve as device handlers and are used by the operating system to make requests of ABIOS. The common data area is made up of two arrays: an array of logical-ID entries and an array of data-pointer entries. Each logical-ID entry contains a pointer to a device block and a pointer to a function transfer table. Each data-pointer entry contains memory addresses that are used by ABIOS services.

On each request to ABIOS, a segment or selector that has an assumed offset of 0 and points to the common data area is passed to ABIOS. This pointer is referred to as the anchor pointer to the common data area.

The following diagram shows the common data area and its relationship to the other ABIOS data structures.

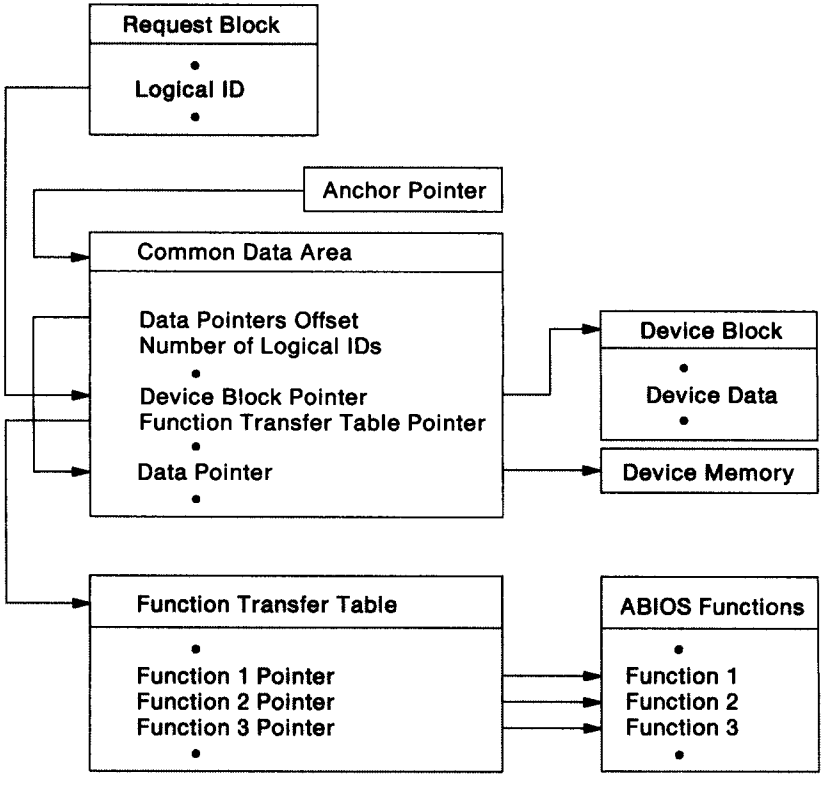


Figure 2-1. Flow of Common Data Area

The following figure shows the format of the common data area.

Size	Offset	Description
Word	00H	Offset to data pointer 0
Word	02H	Count of logical IDs
DWord	04H	Reserved
DWord	08H	Device-block pointer logical ID
DWord	0CH	Function-transfer-table pointer logical ID 1
DWord	10H	Device-block pointer logical ID 2
DWord	14H	Function-transfer-table pointer logical ID 2
.	.	.
.	.	.
.	.	.
DWord	08H×*n*	Device-block pointer logical ID *n*
DWord	(08H×*n*) + 04H	Function-transfer-table pointer logical ID *n*
Word	(08H×*n*) + 08H	Data pointer *p* length
Word	(08H×*n*) + 0AH	Data pointer *p* offset
Word	(08H×*n*) + 0CH	Data pointer *p* segment
Word	(08H×*n*) + 0EH	Data pointer *p*−1 length
Word	(08H×*n*) + 10H	Data pointer *p*−1 offset
Word	(08H×*n*) + 12H	Data pointer *p*−1 segment
.	.	.
.	.	.
.	.	.
Word	(08H×*n*) + (06H×*p*) + 08H	Data pointer 0 length
Word	(08H×*n*) + (06H×*p*) + 0AH	Data pointer 0 offset
Word	(08H×*n*) + (06H×*p*) + 0CH	Data pointer 0 segment
Word	(08H×*n*) + (06H×*p*) + 0EH	Data pointer count

n = the number of logical IDs
p = the number of data pointers minus 1

Figure 2-2. Common Data Area

The common-data-area entries are:

Offset to Data Pointer 0: This field, combined with the anchor pointer, produces a pointer to the Data Pointer 0 Length field.

Count of Logical IDs: This field contains the number of device-block and function-transfer-pointer pairs.

Device-Block Pointers: These fields contain the pointers to the device blocks for the specified logical IDs.

Function-Transfer-Table Pointers: These fields contain the pointers to the function transfer tables for the specified logical IDs.

Data-Pointer Lengths: These fields contain the lengths of the data areas that are pointed to by the associated data pointer.

Data-Pointer Offsets: These fields contain the offsets of the data areas. Each offset is combined with its associated data-pointer segment to produce a pointer to the data area.

Data-Pointer Segments: These fields contain the segments of the data areas. Each segment is combined with its associated data-pointer offset to produce a pointer to the data area.

Data-Pointer Count: This field contains the number of data pointers.

If the Function-Transfer-Table Pointer field and the Device-Block Pointer field are both set to hex 0:0 after ABIOS initialization, the associated logical ID is disregarded by the operating system as a null common-data-area entry. The entry is used as a temporary placeholder during initialization for ABIOS extendibility. For more information, refer to Section 5, "Additional Information."

Function Transfer Table

ABIOS entry points are stored in vector tables, called function transfer tables. These tables contain the doubleword address pointer for each ABIOS function. Reserved function pointers are initialized to hex 0:0. Each logical ID (entry in the common data area) has a function-transfer-table pointer. Multiple logical IDs can have function-transfer-table pointers that point to the same function transfer table.

The following figure shows a function transfer table and its relationship to the common data area.

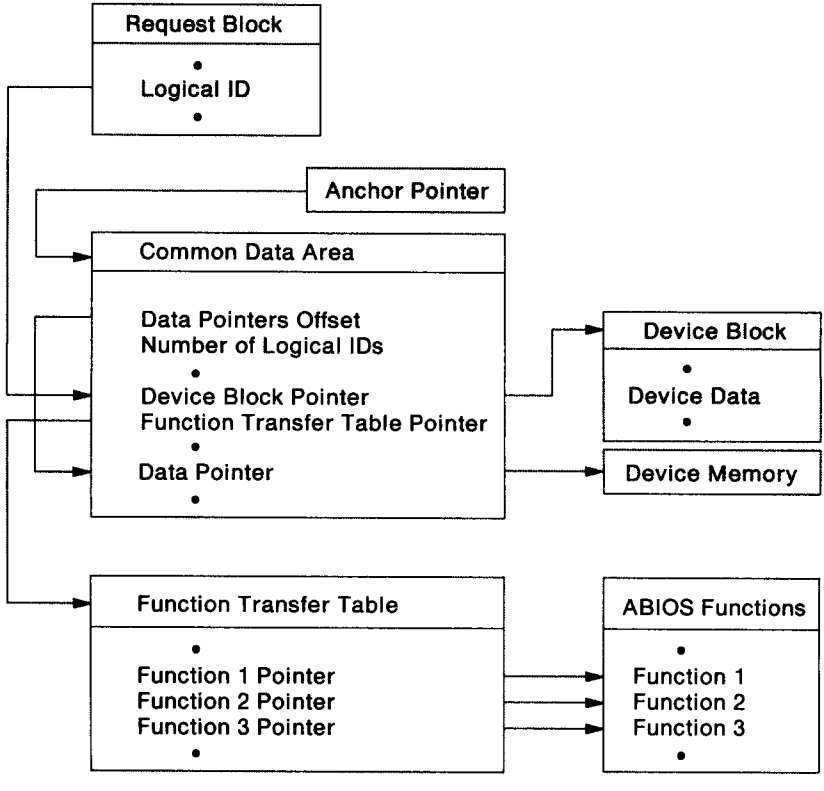


Figure 2-3. Flow of Function Transfer Table

The operating system builds a request block, including the logical ID, which defines the device, and the function. Based on the information in the request block, the function-transfer-table pointer and device-block pointer can be located in the common data area for the requested device. The operating system uses the function-transfer-table pointer to start requests, process interrupts, and handle any time-outs that occur. Each pointer in the function transfer table is a doubleword pointer to a Function routine.

The following figure shows the format of the function transfer table.

Size	Offset	Description
DWord	00H	Start-routine pointer
DWord	04H	Interrupt-routine pointer
DWord	08H	Time-Out routine pointer
Word	0CH	Function count
Word	0EH	Reserved
DWord	10H	Function 1 routine pointer
DWord	14H	Function 2 routine pointer
.	.	.
.	.	.
.	.	.
DWord	(4×*n*) + 0CH	Function *n* routine pointer

n = the number of functions

Figure 2-4. Function Transfer Table

The function-transfer-table entries are:

Start-Routine Pointer: The Start-routine pointer is a doubleword pointer. It is called (using Call Far Indirect) to start a request. This routine validates the Function field, the Request-Block Length field, and the Unit field. All registers are saved and restored across a call to this routine.

Interrupt-Routine Pointer: The Interrupt-routine pointer is a doubleword pointer. It is called (using Call Far Indirect) to resume a multistaged request upon indication from the hardware. All multistaged requests are resumed through this routine if the operation is not complete. All registers are saved and restored across a call to this routine. If this function transfer table corresponds to a device that does not interrupt, the Interrupt-Routine Pointer field is initialized to hex 0:0.

Time-Out Routine Pointer: The Time-Out routine pointer is a doubleword pointer. It is called (using Call Far Indirect) to terminate a request that fails to receive a hardware interrupt within a specified length of time. This routine terminates the request and leaves the hardware controller in a known, initial state. All registers are saved and restored across a call to this routine. If this function transfer table corresponds to a device that does not interrupt, or to a device that interrupts but never times out, the Time-Out Routine Pointer field is initialized to hex 0:0.

Function Count: This is a word count of the number of functions that are supported by a device.

Reserved: This is a reserved word (allocated regardless of whether the value of the Function Count field is 0).

Function 1 Pointer: This is a doubleword pointer to the Function 1 routine.

Function 2 Pointer: This is a doubleword pointer to the Function 2 routine.

Function n Pointer: This is a doubleword pointer to the Function *n* routine.

For more information, see "Functional Parameters" on page 4-5.

Device Block

ABIOS routines require a permanent work area, called a device block, for each device. Hardware port addresses, interrupt levels, and device-status information are stored in a device block.

The device block contains both public and private data. The public data in the device block is a readable area whose format is common across all device blocks. The operating system must not alter this area. Private data in the device block is used internally by ABIOS. Its format and content are not necessarily identical in all implementations of ABIOS. The operating system must not examine or alter private data, and IBM reserves the right to alter the contents of the private portion of the device block.

The following figure shows a device block and its relationship to the common data area.

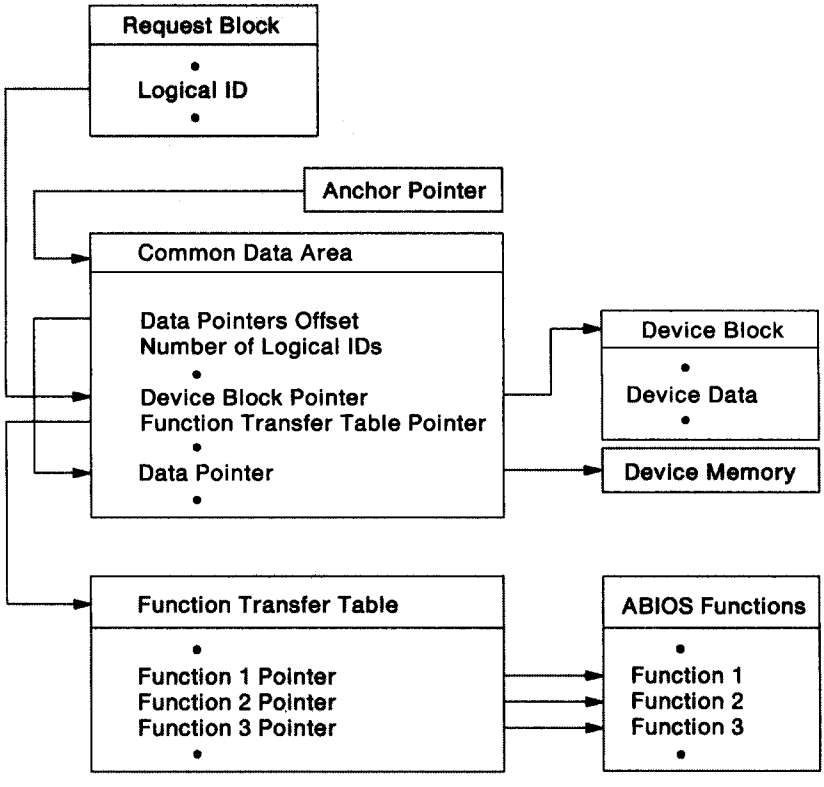


Figure 2-5. Flow of Device Block

Every ABIOS device has an associated device block. The device-block format is shown in the following figure.

Size	Offset	Description	Access
Word	00H	Device-block length	Public read
Byte	02H	Revision	Public read
Byte	03H	Secondary device ID	Public read
Word	04H	Logical ID	Public read
Word	06H	Device ID	Public read
Word	08H	Count of logical-ID exclusive-port pairs	Public read
Word	0AH	Count of logical-ID common-port pairs	Public read
DWord	?	Logical-ID exclusive-port pairs 0	Public read
DWord	?	Logical-ID exclusive-port pairs 1	Public read
.	.	.	
.	.	.	
.	.	.	
DWord	?	Logical-ID exclusive-port pairs *n*	Public read
DWord	?	Logical-ID common-port pairs 0	Public read
DWord	?	Logical-ID common-port pairs 1	Public read
.			
.			
.			
DWord	?	Logical-ID common-port pairs *n*	Public read
Word	?	Device-unique data-area length	Private
?	?	Device-unique data area	Private
Word	?	Count of units	Private
Word	?	Unit-unique data-area length	Private
?	?	Unit-unique data area	Private

? = placeholder for variable values
n = the number of port pairs

Figure 2-6. Device Block

The device-block entries are:

Device-Block Length: This field is one word long, and it contains the number of bytes that are in the device block, including the Device-Block Length field. The maximum specifiable length is 64KB minus 1 byte. The required device-block size for a particular device is returned during ABIOS initialization.

Revision: This byte indicates the level of the supporting code for a device. The initial value of the base level is 0. The value of the Revision field is increased by 1 for each succeeding version of ABIOS code for a particular device ID and secondary device ID (that is, when a new level of ABIOS code is developed for existing hardware).

Secondary Device ID: This byte indicates the level of hardware that an ABIOS implementation supports. The initial value of the base level is 0. The value of the Secondary Device ID field is increased by 1 when a new level of code is developed for a previously-defined device ID that supports new hardware. When the value of the Secondary Device ID field is increased, the Revision field is reset to 0.

Logical ID: This field indicates the logical name of the device that is associated with a device block. It is analogous to the software-interrupt number that BIOS uses to access different device types. Logical ID values are determined dynamically during ABIOS initialization; the logical ID for a given device is determined by the index of its entry in the common data area.

To facilitate the patching of common internal ABIOS functions, the operating system must reserve a number of logical IDs to enable ABIOS to call these common internal ABIOS functions. They are identified to the operating system during ABIOS initialization. The logical-ID values are shown in the following figure.

Logical ID	Usage
00H	Reserved
01H	Reserved
02H to *n*H	ABIOS internal calls
>*n*	System and adapter devices

n = the last logical ID reserved for ABIOS internal calls

Figure 2-7. Logical ID Values

Device ID: This field indicates the type of device that is addressed by a function request and the ABIOS function level that is supported. The assigned values of this field are shown in the following figure.

Device ID	Device
00H	ABIOS Internal Calls
01H	Diskette
02H	Disk
03H	Video
04H	Keyboard
05H	Parallel Port
06H	Asynchronous Communication
07H	System Timer
08H	Real-Time Clock Timer
09H	System Services
0AH	Nonmaskable Interrupt
0BH	Pointing Device
0CH	Reserved
0DH	Reserved
0EH	Nonvolatile Random Access Memory (NVRAM)
0FH	Direct Memory Access (DMA)
10H	Programmable Option Select (POS)
11H to 15H	Reserved
16H	Keyboard Security
17H	SCSI Subsystem Interface
18H	SCSI Peripheral
19H to FFFFH	Reserved

Figure 2-8. Device ID Values

Device ID hex 00 is reserved for ABIOS internal calls (an ABIOS function calling another ABIOS function). Each of the other device-ID values denotes a type of device and a level of ABIOS support, as described in the "Interfaces" section.

Count of Logical-ID Exclusive-Port Pairs: This is the number of logical-ID exclusive-port pairs. A logical-ID exclusive port is a port that is used exclusively by a particular logical ID. Examples are the diskette ports, disk ports, asynchronous communication ports, parallel ports, and video ports. If the value of the Count of Logical-ID Exclusive-Port Pairs field is 0, no space is allocated for the Logical-ID Exclusive-Port Pairs fields.

Count of Logical-ID Common-Port Pairs: This is the number of logical-ID common-port pairs. Logical-ID common ports are ports that are shared across more than one logical-ID value. Examples are the DMA-controller ports, keyboard-controller ports, and NVRAM ports. Each logical ID that uses one of these ports contains an entry in the Logical-ID Common-Port Pairs fields of the device block. If the value of this field is 0, no space is allocated for the Logical-ID Common-Port Pairs fields.

Logical-ID Exclusive-Port Pairs: These are the logical-ID exclusive-port pairs. The first word of the doubleword is the starting I/O-port number of a range of I/O-port numbers. The second word is the ending I/O-port number of the range.

Logical-ID Common-Port Pairs: These are the logical-ID common-port pairs. The first word of the doubleword is the starting I/O-port number of a range of I/O-port numbers. The second word is the ending I/O-port number of the range.

Note: Every port that an ABIOS logical ID reads from or writes to is contained in either the Logical-ID Exclusive-Port Pairs fields or the Logical-ID Common-Port Pairs fields.

Device-Unique Data-Area Length: This field contains the length, in bytes, of the device-unique data area for the specified device.

Device-Unique Data Area: This field contains data that is unique to a device. Parameters that describe the device and working data that span the device ID are kept in this area. This area contains private data for ABIOS; its content and format might change. Examples of the data that is kept in this area are interrupt level, arbitration level, and device status.

Count of Units: This field contains the number of unit-unique data areas in the device block. If the value of this field is 0, the Count of Units field is the last field in the device block.

Unit-Unique Data-Area Length: This field contains the length, in bytes, of a single entry in the repeatable unit-unique data area, excluding the Unit-Unique Data-Area Length field. This field exists only when the value of the Count of Units field is greater than 0.

Unit-Unique Data Area: This field is a private area that is repeatable for each unit of the specified device ID. For example, if the device ID value is hex 01 (diskette), a particular diskette drive is considered a unit. Parameters that describe the unit and working data that span individual requests are kept in this area. This area contains private data for ABIOS; its content and format might change. This field exists only when the value of the Count of Units field is greater than 0.

Section 3. Initialization

© Copyright IBM Corp. 1991

Notes:

Introduction

ABIOS is initialized on demand. The operating system makes specific calls to BIOS and ABIOS to initialize ABIOS. The real-mode common data area must be initialized before any requests can be made to ABIOS. Initialization must be performed in the real mode of the microprocessor. It includes building the system parameters table, the initialization table, and the common data area. The following diagram shows the flow of the real-mode common-data-area initialization.

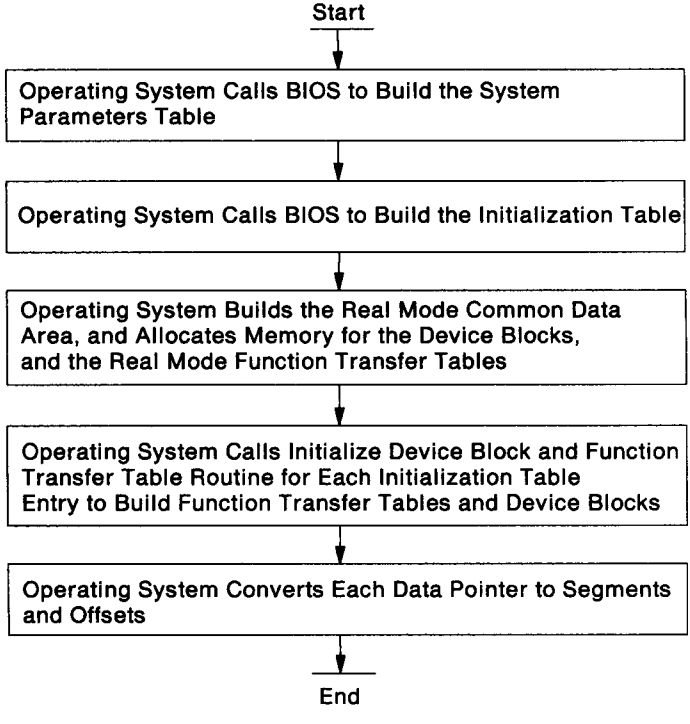


Figure 3-1. Flow of Real-Mode Common-Data-Area Initialization

Build System Parameters Table—Operating System

The operating system allocates an area of hex 20 bytes and calls BIOS to build the system parameters table. This table describes the number of devices that are available in the system, the ABIOS common entry points, and system-stack requirements.

Interrupt 15H—System Services
(AH) = 04H—Build System Parameters Table

Invocation: Software interrupt, operating system calls BIOS

(ES:DI) - Pointer to caller's memory where system parameters table is to be built
(DS) - Segment with assumed offset of hex 0 to RAM extension area (points to a RAM extension with a length of 0 for no RAM extensions)

On Return:
(AH) = 0 - Operation successfully completed
CF = 1 - Exception error

All registers except (AX) and the flags are restored.

Figure 3-2. Build System Parameters Table BIOS Function

Build System Parameters Table—BIOS

When it is called by the operating system, BIOS builds the system parameters table. The Number of Entries field is established from the system-board ROM, adapter ROMs, and RAM extensions. To accumulate the number of entries, configuration information is obtained from system-equipment data areas, from NVRAM, and possibly, by presence testing for devices whose operating code resides in the system-board ROM.

For devices that have code in an adapter ROM, an extension of the power-on self-test (POST) ROM scan determines the number of entries that the adapter ROM requires (see "Adapter-ROM Structure" on page 5-7).

For devices that have code in the RAM-extension area, the RAM-extension scan determines the number of initialization-table entries that the RAM extension requires (see "RAM-Extension Structure" on page 5-9).

When the system-parameters-table information has been obtained, the memory that is allocated for this table can be deallocated and reused by the operating system. The format of the system parameters table is shown below.

Size	Offset	Description
DWord	00H	Common Start routine pointer
DWord	04H	Common Interrupt routine pointer
DWord	08H	Common Time-out routine pointer
Word	0CH	Stack required
DWord	0EH	Reserved
DWord	12H	Reserved
DWord	16H	Reserved
DWord	1AH	Reserved
Word	1EH	Number of entries

Figure 3-3. System Parameters Table

The system-parameters-table entries are:

Common Start Routine Pointer: This is a doubleword address pointer to the Common Start routine entry point.

Common Interrupt Routine Pointer: This is a doubleword address pointer to the Common Interrupt routine entry point.

Common Time-Out Routine Pointer: This is a doubleword address pointer to the Common Time-Out routine entry point.

Stack Required: This field is a word that contains the amount of stack memory, in bytes, that is required for a particular ABIOS implementation.

Number of Entries: This field is a word that contains the number of entries that are required in the initialization table.

Build Initialization Table—Operating System

The initialization table defines the initialization information for each device that the system supports. This information is used to initialize the device blocks and the function transfer tables.

The operating system allocates memory and calls BIOS to build the initialization table. The amount of memory, in bytes, that is required for the initialization table is the number of entries in the initialization table multiplied by hex 18. The Number of Entries field in the system parameters table is used for this calculation. When the initialization process is complete, the memory that was allocated for the initialization table can be deallocated and reused by the operating system.

Interrupt 15H—System Services
(AH) = 05H—Build Initialization Table

Invocation: Software interrupt, operating system calls BIOS

(ES:DI) - Pointer to caller's memory where
initialization table is to be built
(DS) - Segment with assumed offset of hex 0 to
RAM extension area (points to a RAM extension
with a length of 0 for no RAM extensions)

On Return:
(AH) = 0 - Operation successfully completed
CF = 1 - Exception error

All registers except (AX) and the flags are restored.

Figure 3-4. Build Initialization Table BIOS Function

Build Initialization Table—BIOS

BIOS builds the initialization table. This table is established from the system-board ROM, adapter ROMs, and RAM extensions. For devices that have code in an adapter ROM, an extension of the power-on self-test (POST) ROM scan is used. For more information, see "Adapter-ROM Structure" on page 5-7.

For devices that have code in the RAM-extension area, the RAM-extension scan is used (see "RAM-Extension Structure" on page 5-9). All system-board ABIOS-device initialization-table entries precede any adapter-ROM or RAM-extension device entries. The

initialization-table structure, shown in the following figure, is repeated for each entry.

Size	Offset	Description
Word	00H	Device ID
Word	02H	Number of logical IDs
Word	04H	Device-block length
DWord	06H	Initialize Device Block and Function Transfer Table routine pointer
Word	0AH	Request-block length
Word	0CH	Function-transfer-table length
Word	0EH	Data-pointers length
Byte	10H	Secondary device ID
Byte	11H	Revision
Word	12H	Reserved
Word	14H	Reserved
Word	16H	Reserved

Figure 3-5. Initialization Table

The initialization-table entries are:

Device ID: For a list of the values of the Device ID field, see Figure 2-8 on page 2-13. There can be more than one entry in the initialization table with the same device ID.

Number of Logical IDs: This is a word that contains the maximum number of devices that require individual device blocks but are operated by the same code. The Number of Logical IDs field tells the operating system the maximum number of logical IDs that this initialization-table entry allows.

Device-Block Length: This is a word that contains the length, in bytes, of the storage allocation that is required for the device block for this device. A device-block length of 0 indicates that this initialization-table entry is for an ABIOS patch or extension, and no device block needs to be built (see "Adding, Patching, Extending, and Replacing" on page 5-6). When the device-block length is 0, the operating system ensures that the device-block pointer in the common data area is initialized to hex 0:0.

Initialize Device Block and Function Transfer Table Routine Pointer: This is a doubleword address pointer (real mode segment:offset) to the routine to initialize the device blocks and function transfer tables for an entry in the initialization table. This routine is also provided by adapter ROMs or RAM extensions to add, patch, extend, or replace

services (see "Adding, Patching, Extending, and Replacing" on page 5-6).

Request-Block Length: This is a word that contains the length, in bytes, of the storage allocation that is required for the request block for this device. When a request is made to ABIOS, any request-block size that is greater than the returned size is valid.

Function-Transfer-Table Length: This is a word that contains the length, in bytes, of the function transfer table. A function-transfer-table length of 0 indicates that this initialization-table entry is for an ABIOS patch, and no function-transfer-table data area is to be allocated. When the function-transfer-table length is 0, the operating system ensures that the Function-Transfer-Table Pointer field in the common data area is initialized to 0:0.

Data-Pointers Length: This is a word that contains the length, in bytes, of the storage allocation that is required for the Data Pointer fields in the common data area.

Secondary Device ID: This is a byte that is used to determine the level of hardware that an ABIOS implementation supports. See "Device Block" on page 2-9 for more information.

Revision: This byte is used to indicate the level of the supporting code for this device. See "Device Block" on page 2-9 for more information.

Build Common Data Area—Operating System

After the system parameters table and the initialization table are built, the operating system has all the necessary information that is required to build the common data area and its associated data structures (see "Data Structures" on page 1-4). The size of the common data area, the size of each function transfer table, and the size of each device block can be determined from the initialization table.

The operating system builds the common data area at offset hex 00 in a segment and allocates memory for each device block and function transfer table. Memory is allocated in the common data area for the data pointers. The offset to the Data Pointer 0 field is initialized to point to the Data Pointer Length 0 field in the common data area. The Data Pointer Count field is initialized to 0. The Count of Logical IDs

field is filled in with the number of device-block and function-transfer-table pointer pairs. Each device-block pointer and each function-transfer-table pointer is initialized to point to the memory that has been allocated.

Logical-ID values for physical devices are assigned by their order in the initialization table. For example, if the value of the Number of Logical IDs field is 1 for each entry in the initialization table, the first entry corresponds to logical ID 2, the second entry corresponds to logical ID 3, and so on. If the value of the Number of Logical IDs field is greater than 1 for the first initialization-table entry, that entry corresponds to logical ID 2 through logical ID 2 plus the value of the Number of Logical IDs field minus 1. The second initialization-table entry corresponds to the next succeeding logical ID.

Multiple function-transfer-table pointers can point to the same function transfer table. This occurs when the value of the Number of Logical IDs field in an initialization-table entry is greater than 1. The operating system must ensure that the function-transfer-table pointers for the succeeding logical IDs, which correspond to a single initialization-table entry, point to the same function transfer table.

Initialize Pointers—Operating System

The operating system calls the Initialize Device Block and Function Transfer Table routine once for each entry in the initialization table. The operating system passes the anchor pointer, the starting logical ID, and the number of logical IDs that are to be initialized.

The Initialize Device Block and Function Transfer Table routines are called in the order in which their pointers appear in the initialization table. This causes system-board ROM devices to be initialized before any adapter ROM or RAM extension. The Initialize Device Block and Function Transfer Table routines for adapter-ROM devices and RAM extensions can then identify the system-board services that might be needed. This is accomplished by scanning the common data area, using the device ID in the public portion of the device block to identify the system-board service that is needed. When the device ID has been found, the logical-ID number that is in the public portion of the device block is used for all subsequent requests to the system-board ABIOS service.

The operating system needs to call the Initialize Device Block and Function Transfer Table routine only for the devices that are to be made operational. The operating system can determine whether to

initialize an ABIOS device on the basis of the values in the Device ID field and the Secondary Device ID field in each initialization-table entry. For devices that are initialized, the operating system must ensure that each additional initialization-table entry that contains the same device-ID and secondary-device-ID values must also be initialized to allow for patching. Each initialization-table entry that contains a device ID of 0 must be initialized to ensure that internal ABIOS calls are supported. Also, there are device IDs that might need to be initialized to support other device IDs. For example, DMA ABIOS must be initialized if fixed-disk ABIOS is initialized. These requirements are defined in the "Interfaces" section.

When the value of the Number of Logical IDs field is greater than 1, the operating system can initialize any number of logical IDs up to and including the value of the Number of Logical IDs field.

```
Invocation: Call FAR; operating system calls ABIOS on
            system-board ROM, on adapter ROM, or on
            RAM extension, depending on the device.

  (CX) - Number of logical IDs to be initialized (up to
         the value of the Number of Logical IDs field
         in the initialization table)
  (DX) - Starting logical ID
  (DS) - Anchor pointer to the common data area

On Return:
  (AL) - Exception condition
     = 00H - Operation successfully completed
     = 01H to FFH - Device-initialization failure

  All registers except (AX) are restored.
```

Figure 3-6. Initialize Device Block and Function Transfer Table Routine

Initialize Data Structures—ABIOS

When the Initialize Device Block and Function Transfer Table routine is called, ABIOS fills in the function transfer table at the location that is defined by the function-transfer-table pointer of the Starting Logical ID parameter (DX).

For adapter ROMs or RAM extensions, when the Initialize Device Block and Function Transfer Table routine is called, each segment value that is placed in the function transfer table must equal the segment value of its corresponding ROM header or RAM-extension header. This allows an operating system to read the Length field of the ROM header or the RAM-extension header to determine the

segment limit in a bimodal or protected-mode environment. When the protected-mode common data area is being built, if offset hex 00 of the ROM-header segment or RAM-extension-header segment contains the ROM or RAM signature, offset hex 02 contains the length, in multiples of 512 bytes (the limit is hex 7F). This value is used to calculate the segment limit.

After the function transfer table is filled in, the Initialize Device Block and Function Transfer Table routine fills in the device block for the Starting Logical ID parameter (DX) and each succeeding logical ID, up to the value in the Number of Logical IDs to Be Initialized parameter (CX).

On return from the Initialize Device Block and Function Transfer Table routine, if the value in the Exception Condition parameter (AL) is nonzero, indicating an error, deallocate the associated device blocks and function-transfer-table areas and replace the associated device-block pointers and function-transfer-table pointers with hex 0:0, making those entries null common-data-area entries.

Data Pointers

The Initialize Device Block and Function Transfer Table routine stores all the necessary ABIOS data pointers in the data-pointer portion of the common data area. As the data pointers are stored, the value of the Data Pointer Count field is increased. The offset to the stored data pointer in the common data area can be stored in the device block as a handle to the data pointer.

ABIOS initializes data pointers as 32-bit physical addresses that are stored in Intel data format (low word, high word) in the Data Pointer Offset field. Preceding this 32-bit physical address is the Data-Pointer Length field, which indicates the segment limit for a protected-mode or bimodal implementation. In bimodal implementations, a data-pointer value of hex 0:0 in the real-mode version of the common data area indicates an address above 1MB.

The operating system must translate the 32-bit physical address of each data pointer to a 16-bit offset and a 16-bit segment before making any requests to ABIOS.

Logical ID 2 Initialization

Reserved data pointers are initialized by a call to the Initialize Device Block and Function Transfer Table routine for Logical ID 2. Logical ID 2 is reserved for ABIOS internal calls (device ID hex 00).

The following is a list of the reserved data pointers.

Data-Pointer Number	Value (Physical)	Limit	Description
0	400H	0100H	BIOS data area
1	E0000H	FFFFH	First 64KB of system-board ROM
2	F0000H	FFFFH	Second 64KB of system-board ROM

Figure 3-7. Reserved Data Pointers

These data pointers allow a single common data pointer to be used by multiple ABIOS devices instead of duplicating the same data pointer multiple times.

In addition, a call to the Initialize Device Block and Function Transfer Table routine for logical ID 2 places the Common Start routine pointer, the Common Interrupt routine pointer, and the Common Time-Out routine pointer at the Start, Interrupt, and Time-Out pointers in the function transfer table for logical ID 2. The initialization-table entry for the first device of 0 must have a function-transfer-table length of at least hex 10 (greater if ABIOS internal functions exist) to allow for the three doubleword pointers, a word count of functions, and a reserved field.

When the value of the Function Count field in the function transfer table for logical ID 2 is 0, the function transfer table has the following format:

Size	Offset	Description
DWord	00H	Common Start routine pointer
DWord	04H	Common Interrupt routine pointer
DWord	08H	Common Time-Out routine pointer
Word	0CH	Function count (set to 0)
Word	0EH	Reserved

Figure 3-8. Function Transfer Table for Logical ID 2

Build Protected-Mode Tables

For protected-mode or bimodal implementations, it is necessary to build the protected-mode common data area and function transfer tables using the information that is built into the real-mode common data area and function transfer tables. The operating system must create selectors in the protected-mode common data area and function transfer tables whose effective addresses are identical to their corresponding segments in the real-mode common data area and function transfer tables. The following diagram illustrates the necessary steps to build the protected-mode common data area.

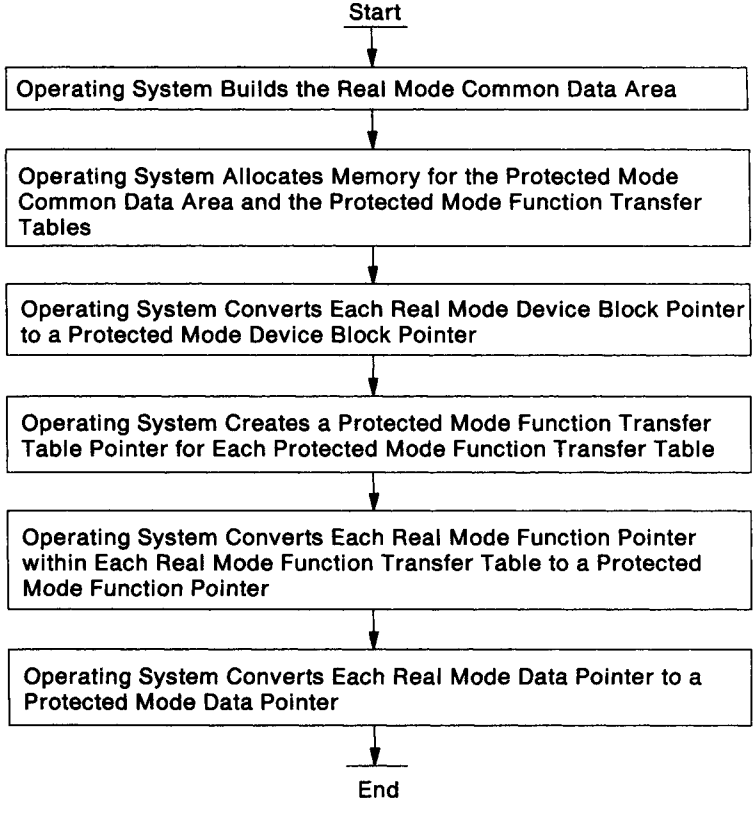


Figure 3-9. Flow of Protected-Mode Common-Data-Area Initialization

To build the descriptors that are associated with each selector, in addition to the physical address, the operating system needs to know the access rights and the segment limit of each segment.

The function-transfer-table pointers and the device-block pointers are writable data-segment descriptors whose expansion direction and limit are maintained by the operating system. The length of each of these tables is returned to the operating system through the Function-Transfer-Table Length field and the Device-Block Length field of the initialization table.

The selector of each data pointer must pertain to a writable data-segment descriptor whose expansion direction is up. The segment limit is determined by the Data Pointers Length field of each data-pointer entry in the common data area.

The pointers to ABIOS functions in the function transfer table must be readable code-segment descriptors whose conforming bit is determined by the operating system. If offset hex 00 of the ROM-header segment or the RAM-extension-header segment contains the ROM or RAM signature, offset hex 02 contains the length, in multiples of 512 bytes (the limit is hex 7F). This value is to be used as the segment limit. If the ROM or RAM signature does not exist, the segment limit is hex FFFF.

If ABIOS is called as a conforming code segment by multiple privilege levels, the operating system is responsible for ensuring that ABIOS has I/O privilege at all times.

Each common-data-area entry in the protected-mode version must be a null common-data-area entry if its corresponding entry in the real-mode version is a null common-data-area entry. When the protected-mode version of each function transfer table is initialized, each entry in the protected-mode version that has a corresponding entry of hex 0:0 in the real-mode version must have a value of hex 0:0 to indicate that the function is not supported. The offset fields in the function transfer table must be the same for the corresponding entries in both tables. The device-block pointers for each logical-ID entry in both the real-mode and the protected-mode common data areas must point to the same device block.

Section 4. Transfer Conventions

© Copyright IBM Corp. 1991

Notes:

ABIOS can be implemented in three environments: protected mode only, real mode only, and bimodal. ABIOS requires a method of transferring control from the caller of ABIOS to ABIOS without sacrificing performance. The two methods that are provided for this transfer are the ABIOS transfer convention and the operating-system transfer convention. Both of these conventions use the request block as the method by which an operating system communicates with and passes parameters to ABIOS.

Request Block

The request block is a parameter block that is used to communicate information bidirectionally between the caller and an ABIOS service. Parameters are passed by the caller (IN) and returned by ABIOS (OUT).

The following diagram shows the request block and its relationship to a common data area.

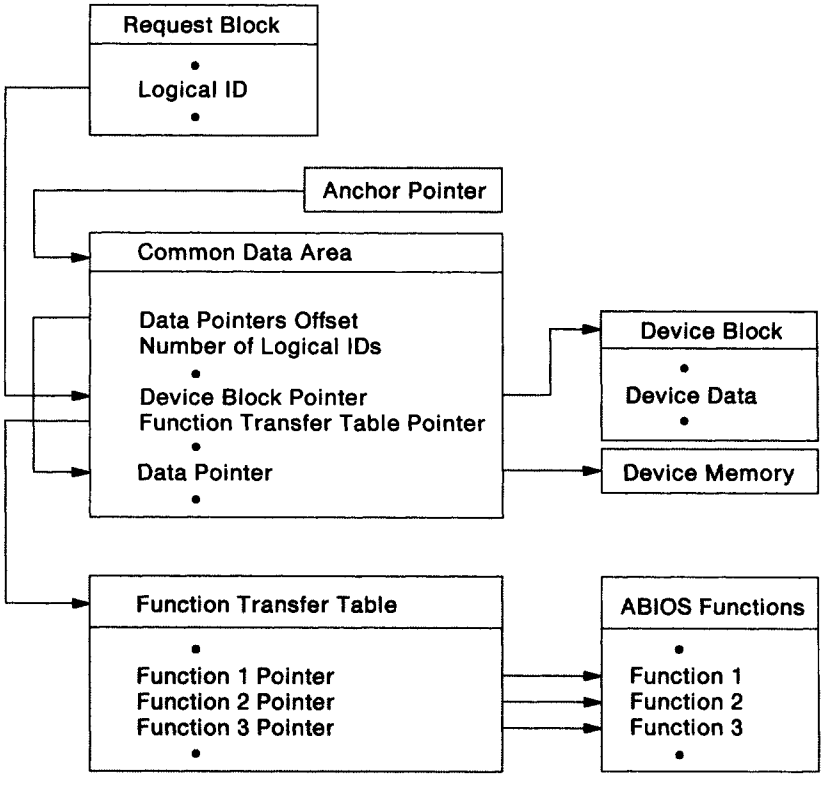


Figure 4-1. Flow of Request Block

Input parameters (IN) are not altered by ABIOS during a request. Output parameters (OUT) and work areas do not need to be set to any predefined values before ABIOS is called. This allows request blocks to be reused after requests are completed, but it requires that any Work Area fields that contain request-state information be initialized by the ABIOS Start routine to the predefined values. Only input (IN) or input/output (IN/OUT) parameters that change between requests need to be initialized before the request block is reused.

All reserved input fields must be set to 0 by the caller of ABIOS.

The parameters are divided into two categories: functional parameters and service-specific parameters.

Functional Parameters

Functional parameters are common to all ABIOS-service requests. They convey information to ABIOS about which service should be invoked on which device. Each input parameter is initialized by the caller, and when it is initialized, it must remain unaltered until the requested operation is complete. The functional parameters are the Request-Block Length field through the Time-Out field, as shown in Figure 4-2.

Service-Specific Parameters

Service-specific parameters are specific to ABIOS requests. The details of the parameters that are passed by the caller and parameters that are returned by ABIOS depend on the service that has been requested. The service-specific parameters are the Data Pointer 1 field through the Work Area field, as shown in Figure 4-2.

Request-Block Structure

The structure of a request block that contains functional parameters and service-specific parameters is shown below.

Functional Parameters:

Word	00H	Request-block length (IN)
Word	02H	Logical ID (IN)
Word	04H	Unit (IN)
Word	06H	Function (IN)
Word	08H	Reserved
Word	0AH	Reserved
Word	0CH	Return code (IN/OUT)
Word	0EH	Time-out (OUT)

Service Specific Parameters:

Word	10H	Reserved
DWord	12H	Data pointer 1 (IN)
Word	16H	Reserved
Word	18H	Reserved
DWord	1AH	Data pointer 2 (IN)
?	1EH	Parameters (IN/OUT)
?	?	Work area

? = undefined initial value

Figure 4-2. Request Block

Request-Block Length (IN): The Request-Block Length field contains the length, in bytes, of the request block, including the Request-Block Length field itself. The maximum specifiable length is 64KB minus 1 byte. The Request-Block Length field contains a fixed value that is initialized by the caller for the specific logical ID. The size of the request block for a logical ID is returned by the Return Logical ID Parameters function (hex 01) when ABIOS is initialized. However, the request block can be larger than the returned size.

Logical ID (IN): The Logical ID field indicates the particular device that is addressed by a function request. It is analogous to a software-interrupt number that is used by BIOS to access different device types.

Unit (IN): The Unit field is a parameter that addresses a particular unit of a device type within a logical ID. The range of valid values is limited by the number of units that are attached to a single controller. The maximum unit number is $n-1$, where n is the number of units that are attached to the controller. The minimum number of units is 1, which causes the value of the Unit field to be 0.

Function (IN): The Function field is a parameter that is used to request a particular category of operation. The assignment of functions is as follows.

Function hex 00—Default Interrupt Handler:

This function is called, with no service-specific parameters, for each logical ID by way of the Interrupt routine. The request block for the default interrupt handler has a fixed length of hex 10 bytes, and the Return Code field is updated on return with hex 0000 (Operation Successfully Completed) or hex 0005 (Not My Interrupt). For more information on the default interrupt handler, see "Default Interrupt Handler" on page 5-5.

Function hex 01—Return Logical ID Parameters:

This is a single-staged function that is common to all ABIOS device IDs. It returns information pertaining to the logical ID. Its request block has a fixed length of hex 20 bytes.

This function returns the following parameters.

Service-Specific Input

Size	Offset	Description
Word	1AH	Reserved
Word	1CH	Reserved
Word	1EH	Reserved

Service-Specific Output

Size	Offset	Description
Word	0CH	Return code
Byte	10H	Hardware interrupt level = FDH - Interrupt level not available = FEH - Special case for NMI = FFH - Noninterrupting logical ID
Byte	11H	Arbitration level = FDH - Arbitration level not available = FEH - Two arbitration levels are available (see offset hex 1C) = FFH - Not applicable
Word	12H	Device ID
Word	14H	Count of units
Word	16H	Logical-ID flags Bits 15 to 6 - Reserved (set to 0) Bit 5 - Address-limited indicator for DMA devices = 0 - Address capability limited to 16MB = 1 - Address capability limited to 4GB Bit 4 - Generic SCSI disk support availability = 0 - Not available = 1 - Available Bit 3 - Overlapped I/O across units = 0 - Not supported = 1 - Supported Bit 2 - 32-bit offset = 0 - Not enabled = 1 - Enabled Bits 1, 0 - Function read/write/additional-data-transfer data-pointer mode = 00 - No read/write/additional-data-transfer functions are supported = 01 - Data pointer 1, logical Data pointer 2, reserved = 10 - Data pointer 1, reserved Data pointer 2, physical = 11 - Data pointer 1, logical Data pointer 2, physical
Word	18H	Request-block length (for functions other than Default Interrupt Handler and Return Logical ID parameters; variable by logical ID)
Byte	1AH	Secondary device ID
Byte	1BH	Revision
Word	1CH	First and second arbitration levels (valid only when the Arbitration Level field is set to hex FE) Bits 7 to 4 - Second arbitration level Bits 3 to 0 - First arbitration level
Word	1EH	Reserved

The logical ID flags contain 2 bits that indicate the mode (physical or logical) of the data pointer for the Read function (hex 08), the Write function (hex 09), and the Additional Data Transfer function (hex 0A). If this parameter indicates that the pointer should be a logical pointer, data pointer 1 is a

logical pointer, and data pointer 2 is reserved. If this parameter indicates that the pointer should be a physical pointer, data pointer 2 is a physical pointer, and data pointer 1 is reserved. If this parameter indicates that both a logical pointer and a physical pointer are to be passed, data pointer 1 is a logical pointer, and data pointer 2 is a physical pointer. If this parameter indicates that neither a logical pointer nor a physical pointer is to be passed, either this logical ID does not support the Read, Write, and Additional Data Transfer functions, or these functions do not require address pointers. In this case, no space is reserved for data pointers in the request block.

Function 02H—Reserved

Function 03H—Read Device Parameters:

Device-specific parameters are returned.

Function 04H—Set Device Parameters:

Device-specific parameters are set.

Function 05H—Reset/Initialize:

The device is put into a known state.

Function 06H—Enable:

The device is enabled for interrupts (not at an interrupt controller).

Function 07H—Disable:

The device is disabled for interrupts (not at an interrupt controller).

Function 08H—Read:

Data is transferred from the device to memory. The data-pointer mode is determined by the Return Logical ID Parameters function (hex 01).

Function 09H—Write:

Data is transferred from memory to the device. The data-pointer mode is determined by the Return Logical ID Parameters function (hex 01).

Function 0AH—Additional Data Transfer:

The data-pointer mode is determined by the Return Logical ID Parameters function (hex 01).

Functions 0BH to FFH—Additional functions as necessary:

The device-specific functions are described in the "Interfaces" section.

Return Code (IN/OUT): This field contains the results of the current stage of the requested operation. For operations that are single staged or in the final stage of a discrete multistaged operation, the Return Code field indicates the results of the entire operation. The return-code values are shown in the following figure.

Return-Code Value	Definition
0000H	Operation Successfully Completed
0001H	Stage on Interrupt
0002H	Stage on Time
0005H	Not My Interrupt, Stage on Interrupt
0009H	Attention, Stage on Interrupt
0081H	Unexpected Interrupt Reset, Stage on Interrupt
8000H	Device in Use, Request Refused
8001H to 8FFFH	Service-Specific Unsuccessful Operation
9000H to 90FFH	Device Error
9100H to 91FFH	Retryable Device Error
9200H to 9FFFH	Device Error
A000H to A0FFH	Time-out Error
A100H to A1FFH	Retryable Time-Out Error
A200H to AFFFH	Time-Out Error
B000H to B0FFH	Device Error with Time-Out
B100H to B1FFH	Retryable Device Error with Time-Out
B200H to BFFFH	Device Error with Time-Out
C000H	Invalid Logical ID
C001H	Invalid Function
C002H	Reserved
C003H	Invalid Unit Number
C004H	Invalid Request-Block Length
C005H to C01FH	Invalid Service-Specific Parameter
C020H to FFFEH	Service-Specific Unsuccessful Operation
FFFFH	Return Code Field Not Valid

Figure 4-3. Return Codes

The bits in the Return Code field are defined in the following figure.

Bit	Definition
15	Unsuccessful operation
14	Parameter error
13	Time-out error
12	Device error
11 to 9	Reserved
8	Retryable error
7	Unexpected interrupt reset
6 to 4	Reserved
3	Attention
2	Not my interrupt
1	Stage on time
0	Stage on interrupt

Notes:
Bits 14 to 8 are defined as above only when bit 15 is set to 1.
Bits 7 to 0 are defined as above only when bit 15 is set to 0.
If all bits are set to 1, the Return Code field is not valid.

Figure 4-4. Return Code Field Bit Definitions

The caller of ABIOS must initialize the Return Code field to hex FFFF (Return Code Field Not Valid) before calling any ABIOS Start routine. If the operating system has an outstanding request block at interrupt time, it first checks the Return Code field. If the value of the Return Code field is hex FFFF (Return Code Field Not Valid), the operating system considers the Return Code field not set and does not attempt to resume this request. The ABIOS routine sets the Return Code field to its appropriate value when the interrupt is expected.

When ABIOS is processing a request that causes a hardware interrupt, interrupts are disabled between the time when a value is written to the Interrupt Enable port and the time when the value of the Return Code field is changed from hex FFFF (Return Code Field Not Valid) to a return-code value with the stage-on-interrupt bit (bit 0) set to 1. After the value of the Return Code field is changed, the interrupt flag is restored to the value that it contained before it was disabled.

When a hardware interrupt occurs, the caller responds only to requests that have a return-code value with the stage-on-interrupt bit (bit 0) set to 1. Outstanding requests with a return-code value of hex FFFF (Return Code Field Not Valid) are not called.

The caller should also maintain a flag that indicates whether a request has completed the Start routine to the point at which the Return Code field is read. This allows for a situation in which an

interrupt occurs after the Return Code field is set to a valid value (not hex FFFF) but before the caller reads the Return Code field. At this point, a Start routine and an Interrupt routine could be operating on the same request block, within different stack frames, making this flag necessary.

Return codes hex 0009 (Attention) and hex 0002 (Stage on Time) need to be tested only by services that require them. Return code hex 0009 indicates that data is available in a service-specific output parameter, although the function is not complete. Return code hex 0002 indicates that the operation is not complete and must be resumed when a specified length of time has elapsed. This length of time is contained in a service-specific output parameter, depending on the service. In addition, return-code values with bit 15 set to 1 are service specific. These values are documented in the "Interfaces" section.

Return code hex 8000 (Device in Use, Request Refused) is used for device serialization. If a logical ID/unit combination is a serially-reusable device, ABIOS returns this return code when there is an outstanding request on the device.

Time-Out (OUT): The Time-Out field contains the expected duration of the requested stage. This is used to determine when an operation has timed out and needs to be reset by the Time-Out routine. The unit of time is 1 second, and the value occupies bits 15 to 3. Bits 2 to 0 of this field are reserved. A value of 0 in this field indicates that the operation has no time-out value. The Time-Out field is valid for return-code values with the stage-on-interrupt bit (bit 0) set to 1.

Data Pointer 1, Data Pointer 2 (IN): If data pointers are required, they are doubleword pointers to I/O-buffer areas for this request. In a bimodal environment, the effective address must be addressable in the current mode of the microprocessor. The address can be a 32-bit physical address for DMA or a segmented address for programmed I/O. The Return Logical ID Parameters function (hex 01) returns a parameter that indicates the mode (physical or logical) of the data pointer for the Read function (hex 08), the Write function (hex 09), and the Additional Data Transfer function (hex 0A). If this parameter indicates that the pointer should be a logical pointer, data pointer 1 is a logical pointer, and data pointer 2 is reserved. If this parameter indicates that the pointer should be a physical pointer, data pointer 2 is a physical pointer, and data pointer 1 is reserved. If this parameter indicates that both a logical pointer and a physical pointer are to be passed, data pointer 1 is a logical pointer, and data pointer 2 is a physical pointer. If this parameter indicates neither a logical pointer nor a physical pointer is to be passed, either this logical ID does not

support the Read, Write, and Additional Data Transfer functions, or these functions do not require address pointers. In this case, no space is reserved for data pointers in the request block.

Parameters (IN/OUT): Parameters communicate operands and, in some cases, the results of ABIOS functions. Parameter requirements vary by device and requested function. Detailed parameter requirements are documented in the "Interfaces" section.

Work Area: Work Area fields are optional data areas that are reserved for ABIOS. No user data can be stored here. The content of these fields varies by the type of request and the particular device routine that is involved. These fields are not required to be initialized to any value. The caller must not alter their content across multistaged requests. Work Area fields are fields that are not defined as service-specific input or service-specific output parameters in the "Interfaces" section.

ABIOS Transfer Convention

The ABIOS transfer convention makes ABIOS responsible for determining the effective address of a particular ABIOS function. ABIOS indexes into the common data area on the basis of the Logical ID field in the request block to access the necessary pointers, including the effective routine (Start, Interrupt, or Time-Out) pointer. The ABIOS transfer convention is the simplest calling sequence for the operating system. The flow of an ABIOS transfer request is shown below.

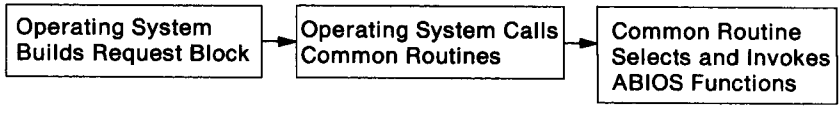


Figure 4-5. Flow of ABIOS Transfer Convention

For this transfer convention, only three routines are available to the caller for transferring control to ABIOS. The pointers to these three routines are returned in the system parameters table when ABIOS is initialized. They are also contained in the function transfer table for logical ID 2. These routines are:

Common Start Routine:

This routine is called (using a Call Far Indirect) to start a request. The Logical ID field in the request block is validated. If this logical-ID value is greater than the value of the Count of Logical IDs field in the common data area, or if this logical-ID value pertains to a null common-data-area entry, the Return Code field is set to hex C000 (Invalid Logical ID).

Common Interrupt Routine:

This routine is called (using a Call Far Indirect) to resume a multistaged request.

Common Time-Out Routine:

This routine is called (using Call Far Indirect) to terminate a request that fails to receive a hardware interrupt within a specified length of time. The Time-Out routine terminates the request and leaves the hardware controller in a known initial state.

The parameter-passing convention for the ABIOS transfer convention is a set of two parameters, two reserved doublewords, and a return address on the stack. The first parameter is the common-data-area

anchor-pointer segment or selector with an assumed offset of hex 00. The second parameter is the doubleword pointer to the request block. The third parameter is a reserved doubleword placeholder for the function-transfer-table pointer. The fourth parameter is a reserved doubleword placeholder for the device-block pointer.

The ABIOS common routines expect the order of the addresses to be from high to low (the order of pushing), as shown in the following figure.

Contents	Displacement from Stack Pointer
Return address of caller	00H
Placeholder for device-block pointer	04H
Placeholder for function-transfer-table pointer	08H
Request-block pointer	0CH
Common-data-area anchor pointer (segment or selector only)	10H

Figure 4-6. ABIOS Transfer Convention Stack Frame

The following pseudocode instructions are suggested:

```
PUSH    Anchor-pointer segment or selector
PUSH    Request-block segment or selector
PUSH    Request-block offset
SUB     Stack pointer, 8
CALL    Common Start routine
```

Pseudocode—ABIOS Transfer Convention

The common routines use the logical ID from the request block and the anchor pointer to determine which device-block pointer and function-transfer-table pointer pair are to be used. These routines take this pair of pointers and place them in the stack-placeholder positions that have been allocated by the caller. Then the common routines transfer control to the Start, Interrupt, or Time-Out routine whose pointers are contained in the function transfer table for the requested value of the Logical ID field. The common-data-area segment or selector, the request-block pointer, the function-transfer-table pointer, and the device-block pointer are passed on the stack. For the ABIOS transfer convention, the caller is responsible for removing the parameters from the stack on return.

The layout of the function transfer table is shown in Figure 2-4 on page 2-8.

Operating-System Transfer Convention

The operating-system transfer convention makes the operating system responsible for determining the effective address of a particular ABIOS function. This method is most useful for handling interrupts from character and programmed-I/O devices that repeatedly call a single routine.

Two methods are available to accomplish operating-system transfers. In the first method, the operating system indexes into the common data area, on the basis of the logical ID, to access the necessary pointers, including the effective routine (Start, Interrupt, or Time-Out) pointer. The advantage of this method over the ABIOS transfer convention is performance.

In the second method, the operating system stores pointers as necessary and accesses them without indexing into the common data area. An operating system might use this method if it is a real-mode-only or protected-mode-only operating system. The common data area is provided to access the necessary pointers as quickly as possible in a bimodal environment. In a single-mode environment, there is no advantage to accessing the pointers by indexing into the common data area. There is a small performance loss with this method. The flow of an operating-system-transfer-convention request is shown below.

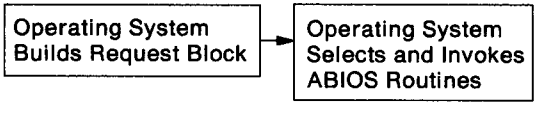


Figure 4-7. Flow of Operating-System Transfer Convention

The parameter-passing convention for the operating-system transfer convention is a set of four parameters and a return address on the stack. The first parameter is the anchor-pointer segment or selector of the common data area, with an assumed offset of hex 00. The second parameter is a doubleword pointer to the request block. The third parameter is a doubleword pointer to the function transfer table. The fourth parameter is a doubleword pointer to the device block.

The Start, Interrupt, and Time-Out routines for each logical ID expect the order of the addresses to be from high to low (the order of pushing), as shown in the following figure.

Contents	Displacement from Stack Pointer
Return address of caller	00H
Device-block pointer	04H
Function-transfer-table pointer	08H
Request-block pointer	0CH
Common-data-area anchor pointer (segment or selector only)	10H

Figure 4-8. Operating-System Transfer Convention Stack Frame

The following pseudocode instructions are suggested:

```
PUSH    Anchor segment or selector
PUSH    Request-block segment or selector
PUSH    Request-block offset
PUSH    Function-transfer-table segment or selector
PUSH    Function-transfer-table offset
PUSH    Device-block segment or selector
PUSH    Device-block offset
CALL    Logical-ID Start routine
```

Pseudocode—Operating-System Transfer Convention

For the operating-system transfer convention, the caller is responsible for removing the parameters from the stack on return.

Section 5. Additional Information

© Copyright IBM Corp. 1991

Notes:

Interrupt Processing

Interrupt Flow

The operating system that communicates with ABIOS provides interrupt handlers that receive control through the hardware interrupt vector. The operating-system interrupt handler must retain the logical IDs of the devices that operate on a specified interrupt level. ABIOS provides routines that are called by the operating-system interrupt handlers.

Each device has a logical ID that is known to the operating system. A logical ID can have one or more active request blocks when an interrupt is processed by the operating-system interrupt handler. Each active request block of the logical ID is processed by calling ABIOS at its interrupt entry point. ABIOS sets the Return Code field to indicate whether the interrupt was associated with the request block.

The operating system can call ABIOS for interrupt processing with interrupts enabled or disabled. ABIOS restores the state of the interrupt flag after any period in which interrupts must be disabled. If no request blocks have the stage-on-interrupt bit (bit 0) of the Return Code field set to 1, and an interrupt occurs, the default interrupt handler is provided to remove the interrupt at the device.

Interrupt Sharing

When more than one logical ID or logical ID/unit combination share an interrupt level, the process is repeated for each logical ID until all logical IDs are processed or the first logical ID with an interrupt is completely processed.

ABIOS expects the operating system to manage End of Interrupt (EOI) processing at the interrupt controller. The method that is used for EOI processing is determined by the operating system. ABIOS does not reset the interrupt controller. The operating system can select its strategy for resetting the interrupt controller after all outstanding request blocks for a particular logical ID are processed through the Interrupt routine and at least one request indicates that the interrupt was serviced. A serviced interrupt request returns from the Interrupt routine with any return-code value other than hex 0005 (Not My Interrupt, Stage on Interrupt).

Rules for Interrupt Processing

One Interrupt Level per Logical ID: Every unit in a particular logical ID operates on the same interrupt level, and no logical ID operates on more than one interrupt level.

One Microprocessor Mode per Call: After being interrupted, ABIOS is returned to the microprocessor mode (real or protected) in which it was running when it was interrupted. That is, after being preempted in the middle of a request stage, it will be returned to the microprocessor mode in which it was running when it was preempted.

Microprocessor-Mode Changes Hidden from ABIOS: While ABIOS function X is running in protected mode, it can be interrupted, and function Y can be invoked in real mode, and vice versa. X can equal Y. After being preempted in the middle of a request stage in one mode, ABIOS can be called through the Start routine in the other mode.

ABIOS Preserves Microprocessor Interrupt Flag State: ABIOS does not change the state of the interrupt flag. ABIOS might temporarily disable the interrupt flag, but it will restore it to its original state. ABIOS never enables the interrupt flag if it is disabled on entry to ABIOS.

Operating System Maintains Request-Block Address Validity: The pointer to a request block that is passed on a request is valid for the duration of that stage of the request.

Data-Area Relocation: The effective memory address of a logical-address pointer (a pointer in the request block in the format "segment:offset" or "selector:offset") can be changed or moved across stages of a request. In the real mode, the segment, the offset, or both can be changed. In the protected mode, the selector, the offset, or both can be changed, and the physical address in the descriptor can be changed.

Operating System Performs EOI: ABIOS does not perform End of Interrupt (EOI) processing on its own behalf. In a level-sensitive interrupt environment, the device condition that causes the interrupt is reset by ABIOS when it processes the request block at the Interrupt routine.

Return Code Indicates Reset of Interrupt Condition: The caller of ABIOS can perform End of Interrupt (EOI) processing when ABIOS returns with a successful return code during processing of the interrupt if all outstanding request blocks for that logical ID have been processed. If the Return Code field contains any value other than hex 0005 (Not My Interrupt, Stage on Interrupt) and all request blocks have been serviced on the logical ID, the caller can assume that the interrupt was serviced (including resetting of the interrupt condition at the device) and process the EOI.

Resetting of Interrupt Condition: Servicing an interrupt for an actual request or for the default interrupt handler resets the interrupting condition at the hardware if the Return Code field contains any value other than hex 0005 (Not My Interrupt, Stage on Interrupt).

Exhaustive Calling: The caller must call ABIOS with each outstanding request per logical ID at interrupt time until the first logical ID with an interrupt is completely processed, which means that each request that has a return-code value with the stage-on-interrupt bit (bit 0) set to 1 for a logical ID has been called.

If multiple outstanding requests per logical ID are waiting for an interrupt, regardless of whether any single request indicates that the interrupt was serviced, each of the requests must be called. This is necessary because resetting the interrupting condition for the first request can reset the interrupt of the second request, causing an interrupt to be lost. Exceptions to this rule are specified in the "Interfaces" section. One exception is the Real-Time Clock Set Interrupt functions (hex 0B, hex 0C, and hex 0F). This cannot happen across logical IDs, because of the following rule concerning interrupts across logical IDs.

Interrupts across Logical IDs: Servicing an interrupt of a given logical ID does not reset the interrupt on another logical ID.

Default Interrupt Handler

In a level-sensitive-interrupt environment, an unexpected hardware interrupt must be handled by resetting the interrupt at the device, as well as at the interrupt controller. ABIOS provides this capability through the use of the default interrupt handler.

Each interrupting ABIOS service provides a default interrupt handler that resets the interrupt at the device and sets the Return Code field to hex 0000 (Operation Successfully Completed) or hex 0005 (Not My Interrupt, Stage on Interrupt). A request block is passed to the default

interrupt handler with no service-specific parameters, and control is transferred to the default interrupt handler through the Interrupt routine. The default interrupt handler is called only if a given logical ID has no outstanding request blocks waiting on interrupt.

To determine whether a logical ID interrupts, call the Return Logical ID Parameters function (hex 01). If the Interrupt Level field pertains to a device that interrupts, it contains the hardware-interrupt level. If the Interrupt Level field contains a value of hex FF, it indicates a noninterrupting logical ID. The nonmaskable interrupt (NMI) device is a special case; it returns a value of hex FE for the interrupt level. If hex FE or hex FF is returned for the interrupt level, the logical ID does not provide a default interrupt handler.

Adding, Patching, Extending, and Replacing

ABIOS provides a mechanism for adding, patching, extending, and replacing the system-board ROM or adapter-ROM ABIOS, using an adapter ROM as well as using RAM. Definitions for adding, patching, extending, and replacing ABIOS are shown below, followed by the mechanisms for accomplishing each.

Adding This adds a previously-unsupported ABIOS interface or adds the support for a new device within the constraints of the old interface, without replacing the old device. An example is adding a new hardware device with ABIOS support. Adding involves a new or old interface, new ABIOS, and new hardware.

Patching This revectors an existing ABIOS function to a patched routine. Patching involves an existing interface, new ABIOS, and existing hardware.

Extending This adds a previously-unsupported function to a particular ABIOS interface that operates on the same device and uses the same device block. Extending involves a new interface, new ABIOS, and existing hardware.

Replacing This involves supporting the existing interface and optionally extending the interface for new hardware of the same device ID. Replacing requires the initialization of a new device block. Replacing involves an existing or new interface, new ABIOS, and new hardware.

The following figure shows these relationships.

	New ABIOS Interface	New ABIOS	New Hardware	New Device Block	New Function Transfer Table
Adding	Yes/No	Yes	Yes	Yes	Yes
Patching	No	Yes	No	No	No
Extending	Yes	Yes	No	No	Yes
Replacing	Yes/No	Yes	Yes	Yes	Yes

Figure 5-1. Adding, Patching, Extending, and Replacing ABIOS

Adapter-ROM Structure

ABIOS provides a facility to integrate adapters with on-board ROM code into the system. During ABIOS initialization, the absolute addresses hex C0000 through hex DF800 are scanned in 2KB blocks to search for a valid adapter ROM.

Adapters that support ROMs can participate in the following convention.

Size	Offset	Description
Word	00H	Signature = hex AA55 (word value)
Byte	02H	Length, in 512-byte blocks
3 bytes	03H	BIOS initialization entry point
Word	06H	Signature = hex BB66 (word value)
Byte	08H	Number of initialization-table entries
–	09H	Build-initialization-table entry point

Figure 5-2. ROM-Module Header

The ROM-module-header entries are:

Signature = Hex AA55 (Word Value): This value in the ROM-module header indicates that this ROM address contains a BIOS ROM, an ABIOS ROM, or both.

Length, in 512-Byte Blocks: This field indicates the length (limit hex 7F) of the ROM that is associated with the ROM-module header.

BIOS Initialization Entry Point: This field is the ROM location that is called by the power-on self-test (POST).

Signature = Hex BB66 (Word Value): This value in the ROM-module header indicates that this ROM address contains an ABIOS ROM.

Number of Initialization-Table Entries: This field contains the number of initialization-table entries that this ABIOS ROM requires. The value of this field for each ABIOS ROM-module header must be at least 1. This field is used to determine the size of the initialization table.

Build-Initialization-Table Entry Point: This field is the location in the ROM of the adapter that Interrupt 15H, Build Initialization Table function ((AH) = 05H, see Figure 3-4 on page 3-6) calls to build the initialization-table entry for the adapter.

The ABIOS structure is similar to the BIOS structure and does not preclude the support of existing adapters that use ROM operating under the BIOS structure. If an adapter ROM is an ABIOS-only adapter ROM, a dummy RETURN FAR instruction must be placed at the BIOS Initialization Entry Point field in the ROM-module header to allow for the BIOS ROM scan during POST.

When the operating system invokes Interrupt 15H, Build System Parameters Table function ((AH) = 04H, see Figure 3-3 on page 3-5), a ROM scan is invoked to determine the number of entries in the initialization table. This number is obtained by accumulating the values in the Number of Initialization-Table Entries field of each ROM-module header and adding that number to the number of entries that are required for the system-board ROM.

When the operating system invokes Interrupt 15H, Build Initialization Table function ((AH) = 05H, see Figure 3-4 on page 3-6), a ROM scan in invoked to search the ROM address space in 2KB increments until a valid ABIOS ROM is detected. The Build Initialization Table Entry function is called for each valid ROM to fill in the initialization table for devices that are operated by the code on the adapter. For more information, see Figure 5-4 on page 5-11.

After the initialization-table entry for the adapter ROM is added to the initialization table, the operating system treats the entry as if it were a system-board entry.

When the Initialize Device Block and Function Transfer Table routine is called for an adapter ROM, each segment value in the function transfer table must equal the segment value of the corresponding ROM-module header.

RAM-Extension Structure

ABIOS provides a facility to integrate adapters with RAM-loadable code into the system. The operating system is responsible for loading the RAM extensions from permanent media to RAM before ABIOS initialization. After ABIOS initialization, RAM extensions can be relocated, but they must always be in memory. During ABIOS initialization, when Interrupt 15H, Build System Parameters Table function ((AH)=04H, see Figure 3-3 on page 3-5) and Build Initialization Table function ((AH)=05H, see Figure 3-4 on page 3-6) are called, a pointer to the RAM-extension area is passed as a parameter.

The layout of a RAM-extension header is shown below.

Size	Offset	Description
Word	00H	Signature = hex AA55 (word value)
Byte	02H	Length, in 512-byte blocks
Byte	03H	Model byte
Byte	04H	Submodel byte
Byte	05H	ROM revision level
Word	06H	Device ID
Byte	08H	Number of initialization-table entries
3 bytes	09H	Build-initialization-table entry point
Byte	0CH	Secondary device ID
Byte	0DH	Revision
Word	0EH	Reserved

Figure 5-3. RAM-Extension Header

The RAM-extension header entries are:

Signature = Hex AA55 (Word Value): This value in the RAM-extension header indicates that this RAM address contains an ABIOS RAM extension.

Length, In 512-Byte Blocks: This field indicates the length (limit hex 7F) of the RAM extension that is associated with the RAM-extension header.

Model Byte, Submodel Byte, ROM Revision Level: These fields describe the system-board ROM with which the RAM extension is associated.

Device ID, Secondary Device ID, Revision: These fields describe the ABIOS service with which the RAM extension is associated.

Number of Initialization-Table Entries: This field contains the number of initialization-table entries that this RAM extension requires. The value of this field for each RAM-extension header must be at least 1. This field is used to determine the size of the initialization table.

Build-Initialization-Table Entry Point: This field contains the location in the RAM extension that Interrupt 15H, Build Initialization Table function ((AH) = 05H, see Figure 3-4 on page 3-6) calls to build the initialization-table entry for this RAM extension.

The RAM-extension area starts on a paragraph boundary and contains a chained list of individual RAM extensions that are linked by way of the Length field. The Reserved fields in the RAM-extension header must be set to 0.

The segment value of each RAM extension is calculated by converting the length of the preceding RAM extension to paragraphs and adding the result to the segment of the preceding RAM extension. If the header for RAM extension 0 is loaded at XXXX:0000 and its length is *n*, meaning that the extension is (*n*/2)KB in length, the header for RAM extension 1 is at location [XXXX + (20H×*n*)]:0000. The last RAM extension in the RAM-extension area points to a RAM extension that has the Length field set to 0.

When the operating system invokes Interrupt 15H, Build System Parameters Table function ((AH) = 04H, see Figure 3-3 on page 3-5), a RAM-extension scan occurs to determine the number of entries in the initialization table. This number is obtained by accumulating the values of the Number of Initialization-Table Entries field in the RAM-extension headers and adding that number to the entries that are required for the system-board and adapter ROMs.

When the operating system invokes Interrupt 15H, Build Initialization Table function ((AH) = 05H, see Figure 3-4 on page 3-6), another RAM-extension scan occurs. The Build Initialization-Table Entry routine (see Figure 5-4 on page 5-11) is called for each RAM extension to fill the initialization-table entry for that RAM extension.

After the initialization-table entry for the RAM extension is added to the initialization table, the operating system treats the entry as if it were a system-board entry.

When the Initialize Device Block and Function Transfer Table routine is called, each segment value in the function transfer table must equal the segment of the corresponding RAM-extension header.

The following figure shows the interface to the Build Initialization Table routine that is used by adapter ROMs and RAM extensions.

Invocation: Call FAR; ABIOS calls adapter ROM or RAM extension strictly for initialization.

(ES:DI) - Pointer to the next available entry in the initialization table

On Return:

(AL) - Exception condition
= 00H - Operation successfully completed
≠ 00H - No entries were added
= 80H - No units were found
(CX) - Number of entries that were added to the initialization table
= 0 - (AL)≠0

All registers except (AX), (CX), and the flags are restored.

Figure 5-4. Build Initialization-Table Entry Routine

Adding

To add a previously-unsupported ABIOS interface, an adapter ROM or RAM extension provides the correct ROM-module or RAM-extension header, and the Build Initialization Table routine is used to build an entry in the initialization table. When the initialization table has been built, it makes no difference to the operating-system initialization process whether the initialization-table entry is associated with a system-board ROM, an adapter ROM, or a RAM extension. The following diagram shows the effect of adding an ABIOS interface.

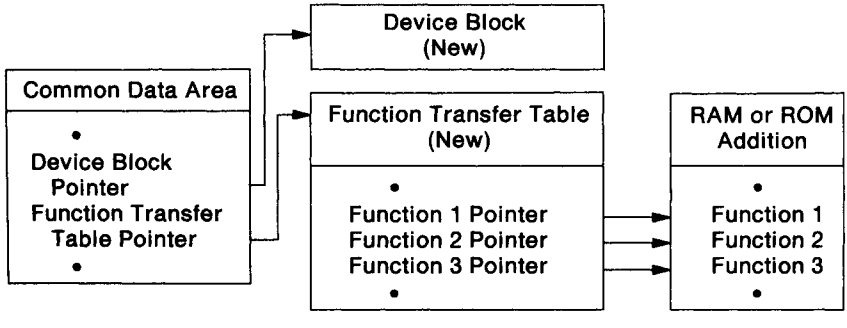


Figure 5-5. Adding ABIOS

Patching

During adapter-ROM scan or RAM-extension scan, an adapter ROM or RAM extension is given control at the Build Initialization Table routine where the adapter ROM or RAM extension builds the new initialization-table entry. When an ABIOS service is patched, the new initialization-table entry that is built is the same as the old initialization-table entry, with the following exceptions:

- The Device-Block Length field is set to 0, indicating that the existing device block suffices for the adapter ROM or RAM extension. Therefore, the operating system should set the Device-Block Pointer field in the common data area that is associated with this initialization-table entry to hex 0:0.
- The Function-Transfer-Table Length field is set to 0, indicating that the existing function transfer table suffices for the adapter ROM or RAM extension. Therefore, the operating system should set the Function Transfer Table Pointer field in the common data area that is associated with this initialization-table entry to hex 0:0.
- The Number of Logical IDs field is set to 1, indicating that this entry requires one logical ID to be initialized for this initialization-table entry.
- The Revision field is set to the value of the Revision field in the old initialization-table entry plus 1.
- The Initialize Device Block and Function Transfer Table Routine Pointer field is initialized to point to the adapter ROM or RAM extension.

When control is transferred to the Initialize Device Block and Function Transfer Table routine, the common data area is scanned for the values of the Device ID field, the Secondary Device ID field, and the Revision field in the device block of the service that is to be patched. This is accomplished by reading the Device-Block Pointer field that is associated with each logical ID and examining the public portion of the device block that contains the Device ID field, the Secondary Device ID field, and the Revision field, until the logical ID (entry in the common data area) that is to be patched is found (see the ABIOS device block in Figure 2-6 on page 2-11). As the common data area is scanned, any null entries should be disregarded. The associated Function-Transfer-Table Pointer field is accessed, and the doubleword pointer of the patched routine is stored at the appropriate offset in the function transfer table.

The Device-Block Pointer field and the Function-Transfer-Table Pointer field that correspond to the Starting Logical ID parameter that is passed to the Initialize Device Block and Function Transfer Table routine are already set to hex 0:0, indicating that the operating system should disregard this entry as a null common-data-area entry.

The following diagram shows the effect of patching an ABIOS interface.

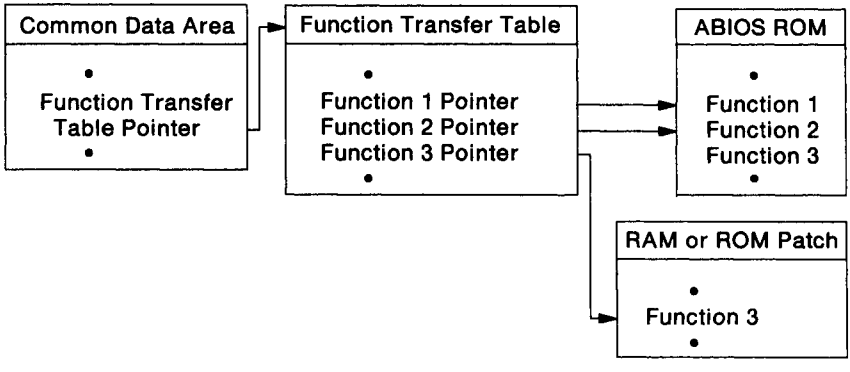


Figure 5-6. Patching ABIOS

Extending

During adapter-ROM scan or RAM-extension scan, an adapter ROM or RAM extension is given control at the Build Initialization Table routine where the adapter ROM or RAM extension builds the new initialization-table entry. When an ABIOS service is extended, the new initialization-table entry that is built is the same as the old initialization-table entry, with the following exceptions:

- The Device-Block Length field is set to 0, indicating that the existing device block suffices for the adapter ROM or RAM extension. Therefore, the operating system should set the Device-Block Pointer field in the common data area that is associated with this initialization-table entry to hex 0:0.
- The Function-Transfer-Table Length field is set to the value of the old Function Transfer Table field plus the length of the extensions.
- The Number of Logical IDs field is set to 1, indicating that this entry requires one logical ID to be initialized for this initialization-table entry.

- The Revision field is set to the value of the Revision field in the old initialization-table entry plus 1.
- The Initialize Device Block and Function Transfer Table Routine Pointer field is initialized to point to the adapter ROM or RAM extension.

When control is transferred to the Initialize Device Block and Function Transfer Table routine, the common data area is scanned for the values of the Device ID field, the Secondary Device ID field, and the Revision field in the device block of the service that is to be extended. This is accomplished by reading the Device-Block Pointer field that is associated with each logical ID and examining the public portion of the device block that contains the Device ID field, the Secondary Device ID field, and the Revision field, until the logical ID (entry in the common data area) that is to be extended is found (see the ABIOS device block in Figure 2-6 on page 2-11). As the common data area is scanned, any null entries should be disregarded. The old function pointers for the service that is to be extended are placed in the new function transfer table, followed by the doubleword pointers to the new functions in the adapter ROM or RAM extension. The Function Count field of the new function transfer table is updated to reflect the number of old functions plus the number of new functions. The Function-Transfer-Table Pointer field in the common data area, which previously pointed to the old function transfer table, is replaced with the pointer to the new function transfer table.

The Device-Block Pointer field that corresponds to the Starting Logical ID parameter that is passed to the Initialize Device Block and Function Transfer Table routine is already set to hex 0:0. The Initialize Device Block and Function Transfer Table routine must set the associated Function-Transfer-Table Pointer field to hex 0:0, indicating a null common-data-area entry.

The following diagram shows the effect of extending an ABIOS interface.

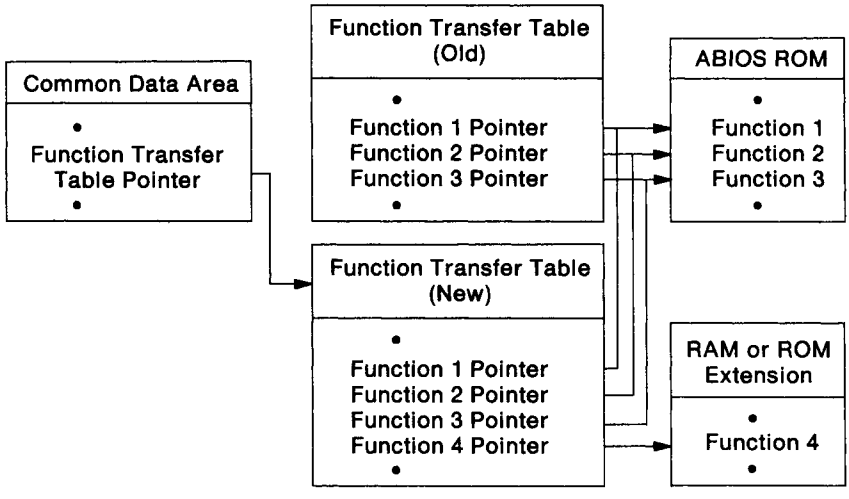


Figure 5-7. Extending ABIOS

Replacing

To replace an ABIOS service, the adapter ROM or RAM extension is given control at the Build Initialization Table routine where the adapter ROM or RAM extension builds the new initialization-table entry. The initialization-table entry is built with a new value in each field except the Device ID field.

When the Initialize Device Block and Function Transfer Table routine is called, the common data area is scanned for the values of the Device ID field, the Secondary Device ID field, and the Revision field in the device block of the service that is to be replaced. This is accomplished by reading the Device-Block Pointer field that is associated with each logical ID and examining the public portion of the device block that contains the Device ID field, the Secondary Device ID field, and the Revision field, until the logical ID (entry in the common data area) that is to be replaced is found (see the ABIOS device block in Figure 2-6 on page 2-11). As the common data area is scanned, any null entries should be disregarded. The new function pointers that point to the adapter ROM or RAM extension are placed in the function transfer table that corresponds to the Starting Logical ID parameter. The Function-Transfer-Table Pointer field in the common data area, which previously pointed to the old function transfer table, is replaced with the pointer to the new function transfer

table. Then the device block is built for the Starting Logical ID parameter. When the device block has been built, the Device-Block Pointer field in the common data area that previously pointed to the old device block is replaced by the pointer to the new device block.

The Initialize Device Block and Function Transfer Table routine must reinitialize the Function-Transfer-Table Pointer field and the Device-Block Pointer field that correspond to the Starting Logical ID parameter to hex 0:0, indicating a null common-data-area entry.

The following diagram shows the effect of replacing an ABIOS interface.

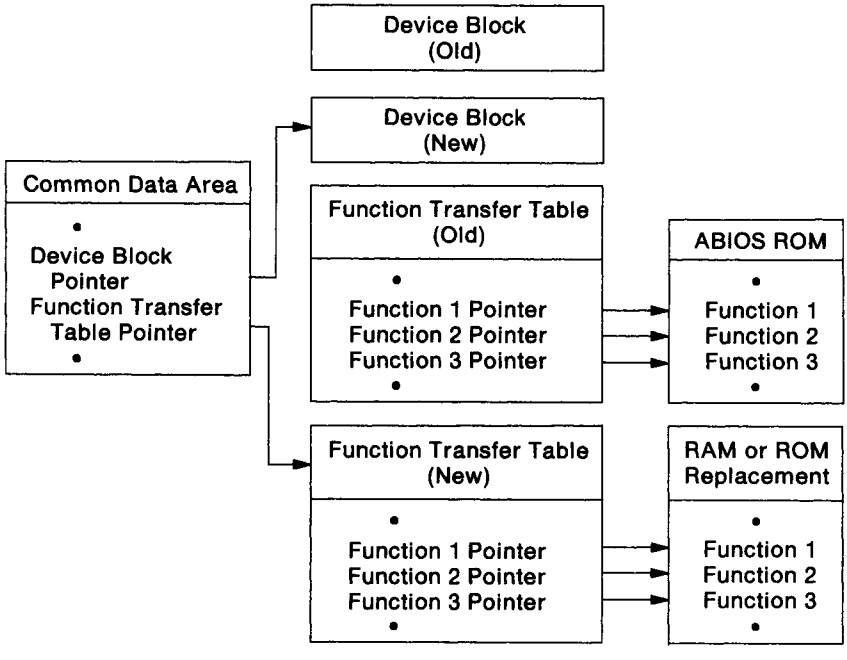


Figure 5-8. Replacing ABIOS

Considerations for RAM Extensions

The Model Byte field, the Submodel Byte field, and the ROM Revision Level field in the RAM-extension header are called system-board identifiers because they describe the system-board ROM that is associated with the RAM extension. The Device ID field, the Secondary Device ID field, and the Revision field are called service identifiers because they describe the specific ABIOS service that is associated with the RAM extension.

Two tests determine whether the RAM extension is required for a particular system:

1. The first test determines whether the RAM extension should remain resident in memory. It must be performed when the RAM extension is loaded. This test determines whether the system-board identifiers in the RAM extension match the system-board identifiers that are returned by Interrupt 15H, Return System Configuration Parameters function ((AH) = C0H). If the system-board identifiers match, the RAM extension remains resident in memory for ABIOS initialization. If the RAM-extension header contains the system-board-identifier wild card (that is, the Model Byte, Submodel Byte, and ROM Revision Level fields are all set to 0), the RAM extension is loaded into memory for all systems.

   To simplify the system-board identifier test, each RAM-extension file must contain RAM extensions with the same system-board identifiers. This allows the system-board-identifier test to be performed against only the first RAM-extension header while it ensures that the test is accurate for all RAM-extension headers in the file.

2. The second test determines whether the service identifiers in the RAM-extension header match a service that exists in the system-board ROM or in an adapter ROM. This test is performed by the Initialize Device Block and Function Transfer Table routine. If a matching service is not found during a scan of the common data area, the Initialize Device Block and Function Transfer Table routine sets the Exception Condition parameter to a nonzero value. When this parameter contains a nonzero value, the operating system makes the associated logical ID a null common-data-area entry.

   If a single RAM extension contains patches for multiple service identifiers, the RAM-extension header must contain the service-identifier wild card (that is, the Device ID field is set to hex 00FF, and the Secondary Device ID and Revision fields are set to hex FF).

For each new version of an ABIOS service that patches, extends, or replaces an existing version, at least one service identifier in the new device block must be different from that in the old device block. The old device block is modified, or a new device block is built by the Initialize Device Block and Function Transfer Table routine, depending on the type of RAM extension (patching, extending, or replacing).

For patching and extending, a new device block is not built; therefore, the Revision field in the existing device block is updated, and the Device ID field and the Secondary Device ID field remain the same. For replacing, a new device block is built because hardware is added; therefore, the Device ID field in the new device block remains the same, but the value of the Secondary Device ID field is increased by 1, and the Revision field is set to 0. For adding, the existing device block is not tested; therefore, the new device block is built as necessary.

The IBM Operating System/2* supports ABIOS updates (RAM extensions) as follows:

- A file called ABIOS.SYS contains a list of file specifications that are separated by blanks or new lines.
- ABIOS.SYS and the files that are associated with the file specifications in ABIOS.SYS are assumed to reside in the root directory of the IPL volume.
- If the RAM extension passes the system-identifier test, the files that are associated with the file specifications in ABIOS.SYS are loaded into memory and appended to one another in the order in which they appear in ABIOS.SYS. These files make up the ABIOS updates that are applied to ABIOS.
- The filename extension must be .BIO, and the sector size of the update files must be a multiple of 512 bytes.

Operating-System Implementation Considerations

ABIOS Rules

The following rules are presented for programmers who are writing operating systems and device drivers.

Rule 1 The operating system must not alter the Device-Block Pointer field, the Function-Transfer-Table Pointer field, or any Data Pointer field for a given logical ID in the common data area during any stage of a request to that logical ID.

* Operating System/2 is a trademark of the International Business Machines Corporation.

Rule 2 After ABIOS is interrupted during any stage of a request, it returns to that stage in the mode in which it was running at the time of the interrupt.

Rule 3 ABIOS device blocks are owned by ABIOS, and only the public portions are accessible by the operating system. There is no guarantee of compatibility of device-block private-area contents across ABIOS implementations.

Rule 4 ABIOS and the operating system share ABIOS request blocks.

Rule 5 ABIOS must traverse the common data area to retrieve the necessary pointers. It does not store pointers in one request or one stage of a request to be used for another request or stage of a request.

Rule 6 ABIOS function X can be interrupted while it is running in protected mode, and function Y can be invoked in real mode, and vice versa. X can equal Y. After ABIOS is preempted in the middle of a request stage in one mode, it can be called through the START routine in the other mode.

Rule 7 ABIOS does not change the state of the interrupt flag. ABIOS can temporarily disable the interrupt flag, but it restores the flag to its original state.

Rule 8 A request-block pointer that is passed on a request is valid for the duration of that stage of the request.

Rule 9 The effective memory address of a physical-address pointer must not be moved for the duration of a single request. When a function requires the data pointer to be passed as a physical address in memory, an external process is assumed to be performing the read or write to memory; therefore, this address cannot change across stages.

Rule 10 The effective memory address of a logical-address pointer (a pointer in the request block in the format "segment:offset" or "selector:offset") can be changed or moved across stages of a request. In real mode, the segment or offset can be changed. In protected mode, the selector or offset can be changed, and the physical address in the descriptor can also be changed.

Rule 11 ABIOS does not perform End of Interrupt processing. In a level-sensitive-interrupt environment, ABIOS resets the device condition that caused the interrupt.

Rule 12 The caller of ABIOS can perform End of Interrupt processing when the Return Code field is set to any value other than hex 0005 (Not My Interrupt, Stage on Interrupt) and all request blocks on the logical ID are serviced. The caller can assume that the interrupt was serviced and process the End of Interrupt.

Rule 13 The caller of ABIOS must call each outstanding request for each logical ID at interrupt time until the first logical ID with an interrupting condition is completely processed. Exceptions are defined in the "Interfaces" section.

Rule 14 The operating system allocates operating-system device numbers on the basis of increasing units within increasing logical IDs. For example, if the first logical ID has a printer device ID, unit 0 is LPT1:, and unit 1 is LPT2:. If unit 1 does not exist and the second logical ID has a printer device ID, unit 0 is LPT2:, and so on.

Rule 15 In a protected-mode or bimodal implementation, ABIOS must have I/O privilege when it is operating in protected mode.

Considerations for Bimodal Implementations

ABIOS is written to be independent of the mode of the microprocessor. Segmented address pointers have different meanings in the two modes, and memory above 1MB is generally not addressable in real mode; therefore, an operating system with a bimodal implementation must conform to the following requirements:

Addressability of Tables: The operating system must ensure that the request blocks, device blocks, function transfer tables, and common data area are always addressable by ABIOS, in the mode in which it is called.

Addressability of Data for Programmed I/O: Non-DMA devices cannot readily use memory above 1MB in real mode. The operating system should allocate the I/O buffers for these devices below 1MB if ABIOS is called in real mode.

Mode Change and Reentrant Routines: When ABIOS is operating in one mode, it can be interrupted and invoked in the other mode. The Interrupt routine and the Time-Out routine are fully reentrant. The Start routines are reentrant with respect to the device blocks. That is, ABIOS can support multiple requests to common code that is operating on different device blocks at the same time within different

stack frames. If the Start routine cannot begin a request, it sets the Return Code field to hex 8000 (Device in Use, Request Refused).

Two Copies of Tables Recommended: Because segmented memory pointers have ambiguous meaning in a bimodal environment, the operating system should keep a real-mode version and a protected-mode version of the common data area and function transfer tables. This avoids the overhead of converting all of the pointers in the tables after switching modes. The offset fields in the function transfer table must be the same for corresponding entries in both versions of the table. When there are two copies of the table, a value of hex 0:0 in the Data Pointer field in the real-mode common data area indicates that the address is above 1MB.

ABIOS is not affected by the existence of more than one table. The protected-mode table does not need to be initialized before ABIOS is called in real mode. However, the protected-mode table must be built before ABIOS is called in protected mode.

The following figure illustrates the ABIOS common data area, function transfer tables, and device blocks in a bimodal environment.

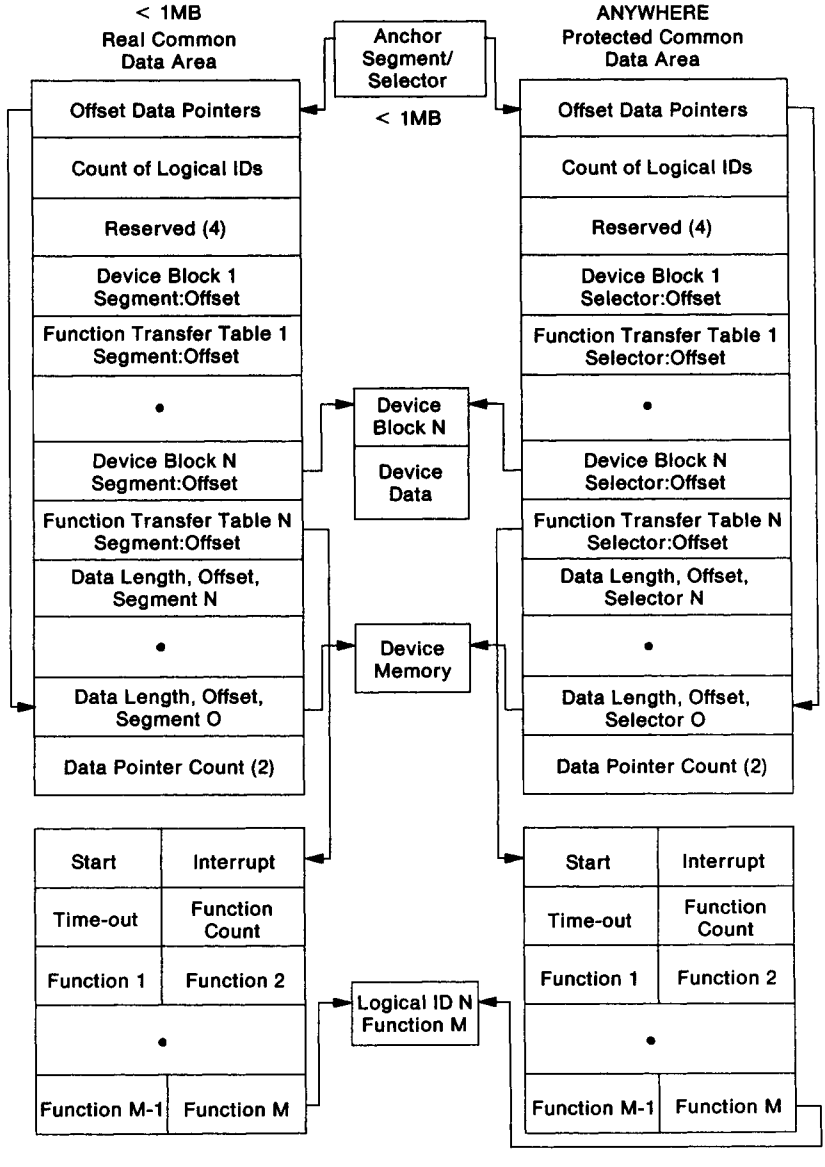


Figure 5-9. Bimodal Data Areas

The fields of the bimodal common data area (Figure 5-9) are:

Anchor Segment/Selector: This is called the anchor pointer. It is a word segment or selector with an assumed offset of hex 0, and it points to the common data area. The segment value that is passed in real mode does not need to equal the selector value that is passed in protected mode.

Real Common Data Area: This is the common data area that is used by ABIOS in the real mode.

Protected Common Data Area: This is the common data area that is used by ABIOS in the protected mode.

Offset Data Pointers: This is an offset that, in conjunction with the anchor pointer, points to the ABIOS Data Pointer 0 Length field.

Count of Logical IDs: This is the number of device-block and function-transfer-table pointer pairs.

Reserved (4): This is a reserved doubleword.

Device Block N Segment:Offset: This is the doubleword pointer to the device block for logical ID *n*.

Function Transfer Table N Segment:Offset: This is the doubleword pointer to the function transfer table for logical ID *n*.

Device Block N Selector:Offset: This is the doubleword pointer to the device block for logical ID *n*.

Function Transfer Table N Selector:Offset: This is the doubleword pointer to the function transfer table for logical ID *n*.

Data Pointer Count (2): This is the number of Data Pointer fields.

Data Length, Offset, Segment N: This is the length, offset, and segment of data pointer *n*.

Data Length, Offset, Selector N: This is the length, offset, and selector of data pointer *n*.

Device Block N: This is the ABIOS device block *n*.

Start: This is a doubleword pointer to the Start routine for this logical ID. It is available to the caller through the operating-system transfer convention.

Interrupt: This is a doubleword pointer to the Interrupt routine for this logical ID. It is available to the caller through the operating-system transfer convention.

Time-Out: This is a doubleword pointer to the Time-Out routine for this logical ID. It is available to the caller through the operating-system transfer convention.

Function Count: This is the number of functions that are supported for this logical ID.

Function M: This is a doubleword pointer to the *m*th Function routine for this logical ID.

Section 6. Interfaces

This section describes the interfaces that are supported by ABIOS. Each interface description includes the interface functions and return-code values. Programming considerations are also included where appropriate.

Parameters are passed to ABIOS functions through request blocks. Input parameters are set by the caller, and output parameters are returned by the ABIOS functions.

This section describes only the service-specific parameters. Functional parameters are described in "Request Block" in the "Transfer Conventions" section.

The following notes apply to each ABIOS device interface in this section:

- The Default Interrupt Handler function (hex 00) and the Logical ID Parameters function (hex 01) are described in "Request Block" in the "Transfer Conventions" section.
- For the Read (hex 08), Write (hex 09), and Additional Data Transfer (hex 0A) functions, the data-pointer mode (physical or logical) should be determined through the Return Logical ID Parameters function (hex 01).
- All reserved input fields must be set to 0.
- All input fields are unaltered by ABIOS across the stages of a request.
- All fields, other than input fields, do not need to be initialized to any predefined values before a request is initiated through the Start routine. The caller must not alter these fields across the stages of a request.
- The following return-code values are returned for parameter errors, although they are not indicated as possible return-code values in each function description:
  - Hex C000 – Invalid Logical ID (ABIOS transfer convention only)
  - Hex C001 – Invalid Function Number
  - Hex C003 – Invalid Unit Number
  - Hex C004 – Invalid Request-Block Length.

© Copyright IBM Corp. 1991

- The return-code value hex 8000 (Device in Use, Request Refused) is used for device serialization. If a logical ID/unit combination is a serially-reusable device, ABIOS returns this value when there is an outstanding request on the device.
- The caller should generically handle the error ranges of the Return Code field as defined for the request block (see the "Transfer Conventions" section). This permits the definition of additional return codes in each of the ranges without affecting the caller's error handling.
- The Time to Wait before Resuming Request field is returned when the Return Code field is set to hex 0002 (Stage on Time).

Device ID 01H—Diskette

Functions

The following are the diskette functions. The Default Interrupt Handler function (hex 00) and the Return Logical ID Parameters function (hex 01) are described in "Request Block" in the "Transfer Conventions" section.

Note: All reserved input fields must be set to 0.

00H—Default Interrupt Handler

01H—Return Logical ID Parameters

02H—Reserved

03H—Read Device Parameters

- This function returns device-control information and the default parameters that are used in diskette operations.
- This function returns bit 6 of the Device Control Flags field to indicate whether the Gap Length for Format field is a required input for the Set Media Type for Format function (hex 0D). If bit 6 is set to 1, the Gap Length for Format parameter is determined on the basis of the Number of Tracks to Be Formatted field and the Number of Sectors per Track field that are passed in the Set Media Type for Format function (hex 0D). If bit 6 is set to 0, the user must provide the Gap Length for Format parameter for the media that is being formatted.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	18H	Reserved

Service-Specific Output

Size	Offset	Description
Word	10H	Number of sectors per track for the maximum media density that is supported by the drive
Word	12H	Size of sector, in bytes = 00H - Reserved = 01H - Reserved = 02H - 512 bytes per sector = 03H to FFFFH - Reserved

Size	Offset	Description
Word	14H	Device control flags Bits 15 to 7 - Reserved Bit 6 - Support of Gap Length for Format parameter for the Set Media Type for Format function (hex 0D) = 0 - User must provide Gap Length for Format parameter = 1 - ABIOS defines the Gap Length for Format parameter on the basis of the Number of Tracks to Be Formatted field and the Number of Sectors per Track field in the Set Media Type for Format function (hex 0D) Bits 5, 4 - Reserved Bit 3 - Recalibration status = 0 - Recalibration is not required = 1 - Recalibration is required Bit 2 - Concurrent operations support = 0 - Not supported = 1 - Supported Bit 1 - Format-unit support = 0 - Not supported = 1 - Supported Bit 0 - Change-signal availability = 0 - Not available = 1 - Available
Word	16H	Diskette drive type = 00H - Drive not present/invalid NVRAM = 01H - 5.25-inch, 40-track, 2-head, 360KB = 02H - 5.25-inch, 80-track, 2-head, 1.2MB = 03H - 3.5-inch, 80-track, 2-head, 720KB = 04H - 3.5 inch, 80-track, 2-head, 1.44MB = 05H - Reserved = 06H - 3.5-inch, 80-track, 2-head, 2.88MB 07H to FFFFH - Reserved
DWord	1CH	Delay before turning off motor (in microseconds)
DWord	20H	Motor-startup time (in microseconds)
Word	26H	Number of cylinders in the maximum media density that is supported by the drive
Byte	2AH	Number of heads
Byte	2BH	Recommended software retry count
Byte	2CH	Fill byte for format
Byte	2DH	Head settle time (in microseconds)
Byte	31H	Gap length for read/write/verify
Byte	32H	Gap length for format
Byte	33H	Data length

04H—Set Device Parameters

- This function can be used to change the default parameters for diskette operations.
- The possible values of the Return Code field are hex 0000, 8000, and C005.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
Word	12H	Sector size, in bytes 00H - Reserved = 01H - Reserved = 02H - 512 bytes per sector = 03H to FFFFH - Reserved
Byte	31H	Gap length for read/write/verify
Byte	33H	Data length

Service-Specific Output

Size	Offset	Description
None		

05H—Reset/Initialize

- This function resets the diskette system to an initial state.
- This function should be issued when switching from BIOS to ABIOS.
- If an error occurs, ABIOS will indicate that a controller reset is required upon entry to the next request.
- The caller is responsible for turning off the motor when the request is completed.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 8000, 9009, 9120, and 9180.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved

Service-Specific Output

Size	Offset	Description
None		

06H—Enable (Reserved)

07H—Disable/Reset Interrupt

- This function resets the interrupt at the device.
- The possible values of the Return Code field are hex 0000, 9120, and 9180.

Service-Specific Input

Size	Offset	Description
Word	18H	Reserved

Service-Specific Output

Size	Offset	Description
None		

08H—Read

- This function transfers data from the specified cylinder, head, and sector on the diskette to the specified memory location. The Return Logical ID Parameters function (hex 01) returns the data-pointer mode (whether it is physical or logical).
- If the 'diskette change' signal is inactive, ABIOS proceeds with the operation.
- If the 'diskette change' signal is active and ABIOS is able to reset the 'diskette change' signal to the inactive state, the Return Code field is set to hex 8006 (Media Changed). However, if the 'diskette change' signal is active and ABIOS is not able to reset the 'diskette change' signal to the inactive state, the Return Code field is set to hex 800D (Media Not Present), and no data is transferred.
- If the Number of Sectors to Be Read field is set to 0, no action is performed, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- ABIOS supports only a block size of 512 bytes per sector.
- The caller is responsible for turning off the motor when the request is completed.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 8000, 8006, 800D, 800E, 9009, 9102, 9103, 9104, 9110, 9120, 9140, 9180, and C00C.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Data pointer 1
Word	16H	Reserved
Word	18H	Reserved
DWord	1AH	Data pointer 2
Word	1EH	Reserved
Word	24H	Number of sectors to be read
Word	26H	Cylinder number (0 based)
Byte	2AH	Head number (0 based)
Word	31H	Sector number

Service-Specific Output

Size	Offset	Description
DWord	20H	Time to wait before resuming request, in microseconds
Word	24H	Number of sectors that were read

09H—Write

- This function transfers data from the specified memory location to the specified cylinder, head, and sector on the diskette.
- If the 'diskette change' signal is inactive, ABIOS proceeds with the operation.
- If the 'diskette change' signal is active and ABIOS is able to reset the 'diskette change' signal to the inactive state, the Return Code field is set to hex 8006 (Media Changed). However, if the 'diskette change' signal is active and ABIOS is not able to reset the 'diskette change' signal to the inactive state, the Return Code field is set to hex 800D (Media Not Present), and no data is transferred.
- If the Number of Sectors to Be Written field is set to 0, no action is performed, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- ABIOS supports only a block size of 512 bytes per sector.
- The caller is responsible for turning off the motor when the request is completed.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 8000, 8003, 8006, 800D, 800E, 9009, 9102, 9104, 9108, 9110, 9120, 9140, 9180, and C00C.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Data pointer 1
Word	16H	Reserved
Word	18H	Reserved
DWord	1AH	Data pointer 2
Word	1EH	Reserved
Word	24H	Number of sectors to be written
Word	26H	Cylinder number (0 based)
Byte	2AH	Head number (0 based)
Word	31H	Sector number

Service-Specific Output

Size	Offset	Description
DWord	20H	Time to wait before resuming request, in microseconds
Word	24H	Number of sectors that were written

0AH—Additional Data Transfer (Subfunction 00H—Format)

- This function writes the field ID from the given buffer for each sector to the specified track.
- Each field ID entry in the buffer is composed of 4 bytes in this order: C, H, R, N, where C is the track number, H is the head number, R is the sector number, and N is the sector size. There must be one entry for every sector on the track.
- Before this function is issued, the Set Media Type for Format function (hex 0D) must be issued once to ensure the proper format parameters.
- The Set Media Type for Format function (hex 0D) must also be issued if the Return Code field is set to hex 8006 (Media Changed) or hex 800D (Media Not Present).
- If the 'diskette change' signal is inactive, ABIOS proceeds with the operation.
- If the 'diskette change' signal is active and ABIOS is able to reset the 'diskette change' signal to the inactive state, the Return Code field is set to hex 8006 (Media Changed). However, if the 'diskette change' signal is active and ABIOS is not able to reset the 'diskette change' signal to the inactive state, the Return Code field is set to hex 800D (Media Not Present), and the field ID is not written.
- ABIOS supports only a block size of 512 bytes per sector.
- The caller is responsible for turning off the motor when the request is completed.

- The possible values of the Return Code field are hex 0000, 0001, 0002, 8000, 8003, 8006, 800D, 800E, 9009, 9102, 9104, 9108, 9110, 9120, 9140, 9180, and C00C.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Data pointer 1
Word	16H	Reserved
Word	18H	Reserved
DWord	1AH	Data pointer 2
Word	1EH	Reserved
Word	24H	Subfunction number
Word	26H	Cylinder number (0 based)
Byte	2AH	Head number (0 based)

Service-Specific Output

Size	Offset	Description
DWord	20H	Time to wait before resuming request, in microseconds

0BH—Verify Sectors

- This function verifies the data on the diskette. The operation is similar to the Read function (hex 08), except that data is not transferred.
- If the 'diskette change' signal is inactive, ABIOS proceeds with the operation.
- If the 'diskette change' signal is active and ABIOS is able to reset the 'diskette change' signal to the inactive state, the Return Code field is set to hex 8006 (Media Changed). However, if the 'diskette change' signal is active and ABIOS is not able to reset the 'diskette change' signal to the inactive state, the Return Code field is set to hex 800D (Media Not Present).
- If the Number of Sectors to Be Verified field is set to 0, no action is performed, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- ABIOS supports only a sector size of 512 bytes per sector.
- The caller is responsible for turning off the motor when the request is completed.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 8000, 8006, 800D, 800E, 9009, 9102, 9104, 9108, 9110, 9120, 9140, 9180, and C00C.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved
Word	1EH	Reserved
Word	24H	Number of sectors to be verified
Word	26H	Cylinder number (0 based)
Byte	2AH	Head number (0 based)
Word	31H	Sector number

Service-Specific Output

Size	Offset	Description
DWord	20H	Time to wait before resuming request, in microseconds
Word	24H	Number of sectors that were verified

0CH—Read Media Parameters

- This function returns the media parameters that were used for the previous operation.
- Because multiple media types might be supported for a single drive type, the Read function (hex 08), Write function (hex 09), Verify Sectors function (hex 0B), or Format function (hex 0A) should be issued to ensure that the proper media parameter values are returned before the Read Media Parameters function (hex 0C) is issued.
- The caller is responsible for turning off the motor when the request is completed.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 8000, 8006, 800D, 9009, 9102, 9104, 9108, 9110, 9120, 9140, 9180, and C00C.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Word	10H	Number of sectors per track
Word	12H	Size of sector, in bytes = 00H - Reserved = 01H - Reserved = 02H - 512 bytes per sector = 03H to FFFFH - Reserved
Word	26H	Number of cylinders
Byte	2AH	Number of heads
Byte	31H	Gap length for read/write/verify
Byte	32H	Gap length for format
Byte	33H	Data length

0DH—Set Media Type for Format

- This function sets the media information for the format operation on the basis of the number of tracks to be formatted (in offset hex 26) and the number of sectors per track (in offset hex 10).
- The presence of media is checked.
  - If the diskette has been removed or the drive door is left open, ABIOS sets the Return Code field to hex 800D (Media Not Present), and the media parameters are not set.
  - If the diskette has been changed and a diskette is present in the drive, ABIOS sets the requested media parameters and resets the 'diskette change' signal to the inactive state.
- If the number of tracks to be formatted (offset hex 26) and the number of sectors per track (offset hex 10) are valid for the supported diskette drive types, ABIOS sets the correct parameters as requested. Otherwise, the Return Code field is set to hex C00C (Unsupported Media Type/Unestablished Media).
- The Read Device Parameters function (hex 03) returns bit 6 of the Device Control Flags field to indicate whether the Gap Length for Format field is a required input for the Set Media Type for Format function (hex 0D). If bit 6 is set to 1, the Gap Length for Format parameter is determined on the basis of the Number of Tracks to Be Formatted field and the Number of Sectors per Track field that are passed in the Set Media Type for Format function (hex 0D). If bit 6 is set to 0, the user must provide the Gap Length for Format parameter for the media that is being formatted.
- This function must be issued once to ensure the proper diskette format information before the Format function (hex 0A) is issued.
- ABIOS uses these parameters until they are changed by the Set Device Parameters function (hex 04) or until the drive door is opened.

- The caller is responsible for turning off the motor when the request is completed.
- The possible values of the Return Code field are hex 0000, 8000, 800D, 800F, C005, and C00C.

Service-Specific Input

Size	Offset	Description
Word	10H	Number of sectors per track
Word	12H	Size of sector, in bytes = 00H - Reserved = 01H - Reserved = 02H - 512 bytes per sector = 03H to FFFFH - Reserved
Word	16H	Reserved
Byte	26H	Number of tracks to be formatted
Byte	2CH	Fill byte for format
Byte	32H	Gap length for format

Service-Specific Output

Size	Offset	Description
DWord	20H	Time to wait before resuming request, in microseconds

0EH—Read 'Diskette Change' Signal Status

- This function returns the state of the 'diskette change' signal. It does not change the state of the 'diskette change' signal.
- The 'Diskette Change' Signal Status field is valid only when the specified drive supports the 'diskette change' signal. The Read Device Parameters function (hex 03) returns information about 'diskette change' signal availability.
- An active 'diskette change' signal indicates that one or more of the following conditions exist:
  - The diskette has been changed.
  - The diskette drive door is open.
  - The diskette-type information is invalid.

  Data is not transferred when the 'diskette change' signal is active.
- The Read function (hex 08), Write function (hex 09), Verify Sectors function (hex 0B), and Format Function (hex 0A) reset the 'diskette change' signal to the inactive state before they begin execution.
- The caller is responsible for turning off the motor when the request is completed.

- The possible values of the Return Code field are hex 0000, 8000, and 800E.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Byte	10H	'Diskette change' signal status = 00H - 'Diskette change' signal is inactive = 01H to 05H - Reserved = 06H - 'Diskette change' signal is active = 07H to FFH - Reserved

0FH—Turn Off Motor

- This function turns the diskette-drive motor off for the requested drive.
- The caller can turn the motor off when the Return Code field is set to hex 0000 (Operation Successfully Completed).
- This function is required for the Reset/Initialize function (hex 05), Read function (hex 08), Write function (hex 09), Additional Data Transfer function (hex 0A), Verify Sectors function (hex 0B), Read Media Parameters function (hex 0C), Set Media Type for Format function (hex 0D), and Read Change Signal Status function (hex 0E).
- The Read Device Parameters function (hex 03) returns the length of the delay before the motor is turned off.
- The possible values of the Return Code field are hex 0000 and 8000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
None		

10H—Interrupt Status

- This function returns the diskette interrupt-pending status. It does not reset the interrupt condition.
- The possible values of the Return Code field are hex 0000 and 8000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Byte	10H	Interrupt-pending status = 00H - No interrupt is pending = 01H - Interrupt pending

11H—Get Media Type

- This function determines the media type that is present in a specified drive.
- If media sense is not supported, the Return Code field is set to hex 8011 (Drive Does Not Support Media Sense). In this case, the media type is considered to be undefined, and control is returned to the caller.
- If no media is found in the selected drive, the Return Code field is set to hex 800D (Media Not Present). In this case, the media type is considered to be undefined, and control is returned to the caller.
- If the drive type of the selected drive does not support the media type that is found in that drive, the Return Code field is set to hex 8010 (Media Type Not Supported by Drive). In this case, the media type that is returned corresponds to the media type that was found.
- If no errors have occurred, the Return Code field is set to hex 0000 (Operation Successfully Completed). In this case, the media type that is returned corresponds to the media type that was found.
- The possible values of the Return Code field are hex 0000, 800D, 8010, and 8011.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Word	10H	Media type found = 00H - Reserved = 03H - Diskette, 1MB (unformatted) = 04H - Diskette, 2MB (unformatted) = 06H - Diskette, 4MB (unformatted) All other values are reserved.

Return Codes

Return codes are returned at offset hex 0C.

Value	Description
0000H	Operation Successfully Completed
0001H	Stage on Interrupt
0002H	Stage on Time
0005H	Not My Interrupt, Stage on Interrupt
8000H	Device Busy, Operation Refused
8003H	Write Attempted on a Write-Protected Diskette
8006H	Media Changed
800DH	Media Not Present
800EH	Change Signal Not Available
800FH	Invalid Value in NVRAM
8010H	Media Type Not Supported by Drive
8011H	Drive Does Not Support Media Sense
9009H	Controller Failure in Reset Operation
9102H	Address Mark Not Found
9104H	Requested Sector Not Found
9108H	DMA Overrun on Operation
9110H	Bad CRC on Diskette Read
9120H	Controller Failure
9140H	Seek Operation Failure
9180H	General Error
A120H	Controller Failure
B020H	Controller Failure
C000H	Invalid Logical ID (ABIOS transfer convention only)
C001H	Invalid Function
C003H	Invalid Unit Number
C004H	Invalid Request Block Length
C005H	Invalid Diskette Parameter
C00CH	Unsupported Media Type/Unestablished Media

Figure 6-1. Diskette Return Codes

Programming Considerations

- Diskette ABIOS indicates in the Return Code field whether an unsuccessful operation needs to be retried. The Read Device Parameters function (hex 03) returns the recommended retry count.
- When the Return Code field is set to hex 0000 (Operation Successfully Completed) the caller can turn off the motor by using the Turn Off Motor function (hex 0F). This function is required for the Reset/Initialize function (hex 05), Read function (hex 08), Write function (hex 09), Additional Data Transfer function (hex 0A), Verify Sectors function (hex 0B), Read Media Parameters function (hex 0C), Set Media Type for Format function (hex 0D), and Read Change Signal Status function (hex 0E). When the request is completed, the caller is responsible for turning off the motor by using the Turn Off Motor function (hex 0F).
- Diskette ABIOS supports crossing of track boundaries, but only switching from head 0 to head 1 on the same cylinder. It does not support switching from head 1 of one cylinder to head 0 of the next cylinder.
- When diskette ABIOS and diskette BIOS requests are issued, the following rules must be followed:
  - Do not attempt an ABIOS call while there is an outstanding BIOS call.
  - Do not attempt a BIOS call while there is an outstanding ABIOS call.
  - The Reset/Initialize function (hex 05) must be the first ABIOS request that follows a BIOS request.
  - The Reset Diskette System BIOS function (Interrupt 13H, (AH) = 00H) must be the first BIOS request that follows an ABIOS request. Also, before any diskette function is issued, bit 4 must be set to 0 in BIOS data area hex 40:90 for drive A and in BIOS data area hex 40:91 for drive B.
  - The Reset/Initialize function (hex 05) must be issued after ABIOS initialization has been completed.
- If an error occurs, ABIOS resets the diskette system.
- The Read function (hex 08), Write function (hex 09), Verify Sectors function (hex 0B), Additional Data Transfer function (hex 0A), and Set Media Type for Format function (hex 0D) reset the 'diskette change' signal to the inactive state before they begin execution.

Diskette Drive Parameters

The following tables list the recommended parameters for diskette drives that are supported on Personal System/2 products.

Byte Definition	320K Media	360K Media
First specification byte	D0H	D0H
Second specification byte	02H	02H
Motor-off time	25H	25H
Bytes per sector	02H	02H
Sectors per track	08H	09H
Gap length	2AH	2AH
Data length	FFH	FFH
Gap length (format)	50H	50H
Fill byte (format)	F6H	F6H
Head settle time (in microseconds)	0FH	0FH
Motor start (in ⅛-seconds)	06H	06H
Maximum track numbers	27H	27H
Data-transfer rate	80H	80H
Multi-rate capability	00H	00H

Figure 6-2. Media Parameter Table – 360KB Slimline Drive

Byte Definition	720K Media
First specification byte	D0H
Second specification byte	02H
Motor-off time	25H
Bytes per sector	02H
Sectors per track	09H
Gap length	2AH
Data length	FFH
Gap length (format)	50H
Fill byte (format)	F6H
Head settle time (in microseconds)	0FH
Motor start (in ⅛-seconds)	04H
Maximum track numbers	4FH
Data-transfer rate	80H
Multi-rate capability	00H

Figure 6-3. Media Parameter Table – 720KB Slimline Drive

Byte Definition	320K Media	360K Media	1.2M Media
First specification byte	E0H	E0H	D0H
Second specification byte	02H	02H	02H
Motor-off time	25H	25H	25H
Bytes per sector	02H	02H	02H
Sectors per track	08H	09H	0FH
Gap length	2AH	2AH	1BH
Data length	FFH	FFH	FFH
Gap length (format)	50H	50H	54H
Fill byte (format)	F6H	F6H	F6H
Head settle time (in microseconds)	0FH	0FH	0FH
Motor start (in ⅛-seconds)	04H	04H	04H
Maximum track numbers	27H	27H	4FH
Data-transfer rate	40H	40H	00H
Multi-rate capability	02H	02H	02H

Figure 6-4. Media Parameter Table – 1.2MB Slimline Drive

Byte Definition	720K Media	1.44M Media
First specification byte	E0H	D0H
Second specification byte	02H	02H
Motor-off time	25H	25H
Bytes per sector	02H	02H
Sectors per track	09H	12H
Gap length	2AH	1BH
Data length	FFH	FFH
Gap length (format)	50H	65H
Fill byte (format)	F6H	F6H
Head settle time (in microseconds)	0FH	0FH
Motor start (in ⅛-seconds)	04H	04H
Maximum track numbers	4FH	4FH
Data-transfer rate	80H	00H
Multi-rate capability	02H	02H

Figure 6-5. Media Parameter Table – 1.44MB Slimline Drive

Byte Definition	720K Media	1.44M Media	2.88M Media
First specification byte	E0H	D0H	A0H
Second specification byte	02H	02H	02H
Motor-off time	25H	25H	25H
Bytes per sector	02H	02H	02H
Sectors per track	09H	12H	24H
Gap length	2AH	1BH	38H
Data length	FFH	FFH	FFH
Gap length (format)	50H	65H	53H
Fill byte (format)	F6H	F6H	F6H
Head settle time (in microseconds)	0FH	0FH	0FH
Motor start (in ⅛-seconds)	04H	04H	04H
Maximum track numbers	4FH	4FH	4FH
Data-transfer rate	80H	00H	C0H
Multi-rate capability	02H	02H	02H

Figure 6-6. Media Parameter Table – 2.88MB Slimline Drive

Byte Definition	720K Media	1.44M Media
First specification byte	D0H	A0H
Second specification byte	02H	02H
Motor-off time	25H	25H
Bytes per sector	02H	02H
Sectors per track	09H	12H
Gap length	2AH	1BH
Data length	FFH	FFH
Gap length (format)	50H	65H
Fill byte (format)	F6H	F6H
Head settle time (in microseconds)	0FH	0FH
Motor start (in ⅛-seconds)	04H	04H
Maximum track numbers	4FH	4FH
Data-transfer rate	80H	00H
Multi-rate capability	02H	02H

Figure 6-7. Media Parameter Table – 1.44MB Half-High Drive

Notes:

Device ID 02H—Fixed Disk

Functions

The following are the fixed disk functions. The Default Interrupt Handler function (hex 00) and the Return Logical ID Parameters function (hex 01) are described in "Request Block" in the "Transfer Conventions" section.

Note: All reserved input fields must be set to 0.

00H—Default Interrupt Handler

01H—Return Logical ID Parameters

02H—Reserved

03H—Read Device Parameters

- This function returns disk-drive information for devices that are specified in the Unit field.
- The possible values of the Return Code field are hex 0000 and 8003.

Service-Specific Input

Size	Offset	Description
Word	28H	Reserved

Service-Specific Output

Size	Offset	Description
Word	10H	Sectors per track associated with unit in request block
Word	12H	Size of sectors in bytes = 00H to 01H - Reserved = 02H - 512-byte sectors = 03H to FFFFH - Reserved

Size	Offset	Description
Word	14H	Device control flags (see "Device Control Flags" on page 6-ID02-18) Bit 15 - SCB transfer support = 1 - SCB transfer function is supported Bit 14 - SCSI Device = 1 - Drive is a SCSI device Bit 13 - Reserved Bits 12, 11 - Format support (values in binary) = 00 - Format is not supported = 01 - Format Track is supported = 10 - Format Unit is supported = 11 - Format Track and Format Unit are supported Bit 10 - ST506 drive = 1 - Drive is ST506 device Bit 9 - Concurrent unit requests per logical ID = 0 - Not concurrent = 1 - Concurrent Bit 8 - Ejecting capability = 0 - Not ejectable = 1 - Ejectable Bit 7 - Media organization = 0 - Random = 1 - Sequential Bit 6 - Locking capability = 1 - Locking is supported Bit 5 - Read capability = 1 - Readable Bit 4 - Caching support = 1 - Caching is supported Bit 3 - Write frequency = 0 - Write once = 1 - Write many Bit 2 - Change-signal support = 1 - Signal is supported Bit 1 - Power is on or off = 1 - Power off Bit 0 - Parameters are valid or not valid = 1 - Parameters not valid
Byte	16H	Logical unit number (LUN); supported only if SCSI device
DWord	18H	Physical number of cylinders that are associated with the unit in the request block
Byte	1CH	Physical number of heads that are associated with the unit in the request block
Byte	1DH	Suggested number of software retries for retryable operations
DWord	20H	Physical number of relative block addresses that are associated with the unit in the request block
DWord	24H	Reserved
Word	28H	Reserved
Word	2CH	Maximum number of blocks to be transferred per one call

04H—Set Device Parameters (Reserved)

05H—Reset/Initialize

- This function resets the disk system to an initial state.
- All return-code values are possible.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	28H	Time to wait before resuming request, in microseconds

06H—Enable (Reserved)

07H—Disable (Reserved)

08H—Read

- The Read function transfers data from the specified relative block address to the specified memory location. The Number of Blocks to Be Read field contains the amount of the data that is to be transferred.
- If the Number of Blocks to Be Read field is set to 0, no action is performed.
- If the value in the Number of Blocks to Be Read field is greater than the maximum number of blocks, no action is performed, and the Return Code field is set to hex C005 (Invalid Count Value).
- The Number of Blocks Read field contains the amount of the data that was transferred.
- When a parameter error is returned, the Number of Blocks Read field is not updated. Also, when a hex 8000, 8001, 8002, 8003, or 800F error is returned, the Number of Blocks Read field is not updated.
- All return-code values are possible.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Data pointer 1
Word	16H	Reserved
Word	18H	Reserved
DWord	1EH	Data pointer 2
Word	1EH	Reserved
DWord	20H	Relative block address
DWord	24H	Reserved
Word	2CH	Number of blocks to be read
Byte	2EH	Flags Bits 7 to 1 - Reserved (set to 0) Bit 0 - Caching = 0 - Caching is OK for this request = 1 - Do not cache on this request

Service-Specific Output

Size	Offset	Description
DWord	28H	Time to wait before resuming request, in microseconds
Word	2CH	Number of blocks read
Word	2FH	Soft-error indicator = 00H - Soft error did not occur ≠ 00H - Soft error occurred

09H—Write

- The Write function transfers data from the specified memory location to the specified relative block address. The Number of Blocks to Be Written field contains the amount of the data that is to be transferred.
- If the Number of Blocks to Be Written field is set to 0, no action is performed.
- If the value in the Number of Blocks to Be Written field is greater than the maximum number of blocks, no action is performed, and the Return Code field is set to hex C005 (Invalid Count Value).
- The Number of Blocks Written field contains the amount of the data that was transferred.
- When a parameter error is returned, the Number of Blocks Written field is not updated. Also, when a hex 8000, 8001, 8002, 8003, or 800F error is returned, the Number of Blocks Written field is not updated.
- All return-code values are possible.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Data pointer 1
Word	16H	Reserved
Word	18H	Reserved
DWord	1AH	Data pointer 2
Word	1EH	Reserved
DWord	20H	Relative block address
DWord	24H	Reserved
Word	2CH	Number of blocks to be written
Byte	2EH	Flags Bits 7 to 1 - Reserved (set to 0) Bit 0 - Caching = 0 - Caching is OK for this request = 1 - Do not cache on this request

Service-Specific Output

Size	Offset	Description
DWord	28H	Time to wait before resuming request, in microseconds
Word	2CH	Number of blocks written
Word	2FH	Soft-error indicator = 00H - Soft error did not occur ≠ 00H - Soft error occurred

0AH—Write Verify

- The Write Verify function operates similarly to the Write function with the addition of a Read Verify function.
- If the Number of Blocks to Be Written/Verified field is set to 0, no action is performed.
- If the value in the Number of Blocks to Be Written/Verified field is greater than the maximum number of blocks, no action is performed, and the return code field is set to hex C005 (Invalid Count Value).
- The Number of Blocks Written field contains the amount of the data that was transferred.
- When a parameter error is returned, the Number of Blocks Written field is not updated. Also, when a hex 8000, 8001, 8002, 8003, or 800F error is returned, the Number of Blocks Written field is not updated.
- All return-code values are possible.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Data pointer 1
Word	16H	Reserved
Word	18H	Reserved
DWord	1AH	Data pointer 2
Word	1EH	Reserved
DWord	20H	Relative block address
DWord	24H	Reserved
Word	2CH	Number of blocks to be written/verified
Byte	2EH	Flags Bits 7 to 1 - Reserved (set to 0) Bit 0 - Caching = 0 - Caching is OK for this request = 1 - Do not cache on this request
Word	31H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	28H	Time to wait before resuming request, in microseconds
Word	2CH	Number of blocks written
Word	2FH	Soft-error indicator = 00H - Soft error did not occur ≠ 00H - Soft error occurred

0BH—Verify

- The Verify function reads from the specified relative block address without transferring any data to system memory. This function verifies the readability of the data.
- If the Number of Blocks to Be Verified field is set to 0, no action is performed.
- If the value in the Number of Blocks to Be Verified field is greater than the maximum number of blocks, no action is performed, and the Return Code field is set to hex C005 (Invalid Count Value).
- All return-code values are possible.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved
Word	18H	Reserved
Word	1EH	Reserved
DWord	20H	Relative block address
DWord	24H	Reserved
Word	2CH	Number of blocks to be verified

Service-Specific Output

Size	Offset	Description
DWord	28H	Time to wait before resuming request, in microseconds
Word	2FH	Soft-error indicator = 00H - Soft error did not occur ≠ 00H - Soft error occurred

0CH—Interrupt Status

- This function returns the disk-interrupt-pending status. It does not reset the interrupt condition.
- After ABIOS checks the Unit field for validity, the Unit field is not used in determining the interrupt status. The interrupt status reports that an interrupt is pending on the logical ID, and it might or might not be the Unit field of the request block.
- If there is a parameter error, the Interrupt Status field is undefined.
- The possible values of the Return Code field are hex 0000 and 8003.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Byte	10H	Interrupt status = 00H - Interrupt not pending = 01H - Interrupt pending = 02H to FFH - Reserved

10H—Set DMA Pacing Factor

- This function sets the DMA pacing for the adapter of the specified drive. All devices that are attached to this adapter are also affected.
- The pacing value is expressed as a percentage; valid values are from 25 to 100, inclusive.
- BIOS does not check the range of the pacing value. Using a value outside the specified range might cause an adapter error.
- No commands can be pending when the request is made.
- All return-code values are possible.

Service-Specific Input

Size	Offset	Description
Byte	10H	Pacing value; valid values are from 25% to 100%
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	28H	Time to wait before resuming request, in microseconds

11H—Return DMA Pacing Factor

- This function returns the current pacing value of the adapter for the specified drive.
- The pacing value is expressed as a percentage; valid values are from 25% to 100%, inclusive.
- The possible values of the Return Code field are hex 0000 and 8003.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Byte	10H	Current pacing value

12H—Transfer SCB

- This function programs the adapter to process the subsystem control block (SCB) that is pointed to by the Physical Pointer to SCB field. (See the technical reference manual for the SCSI Adapter.)
- The SCB-transfer-support bit (bit 15 of the Device Control Flags field) indicates whether this function is supported.
- ABIOS does not check the validity of the SCB.
- If the adapter reports an error, ABIOS determines whether a termination status block (TSB) was returned. ABIOS evaluates the TSB and returns the appropriate error code and the pointer to the failing SCB header. Check bit 0 of the Status field to determine whether the Logical Pointer to Chain Header field (offset hex 1E on output) is valid.

  The SCB chain header has the following format:

Size	Offset	Description
Word	00H	Reserved
DWord	02H	Logical pointer to next SCB header in chain, or chain-ending indicator (0)
Word	06H	Reserved
Word	08H	Reserved
DWord	0AH	Logical pointer to TSB that is associated with this SCB
Word	0EH	Reserved

- The TSB for each SCB in the chain is examined to determine which SCB caused the error; bit 0 in word 0 of the TSB is used to determine whether the error occurred for that SCB. Therefore, the caller must ensure that one of the following occurs:
  - A TSB is returned for each SCB (which can degrade system performance).
  - A TSB is returned only on an error. In this case, the controlling program must ensure that bit 0 of word 0 in the TSB is set to 1 before ABIOS is invoked.
- A logical pointer of 0 ends the SCB chain.
- The chain must have an ending.
- Chains that are passed to BIOS are not dynamically updated.
- If the Logical Pointer to SCB Chain Header field (offset hex 16 on input) is set to 0, ABIOS does not initiate the SCB transfer, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- The chain header must immediately precede the SCB.

- All return-code values are possible.

See "Transfer SCB Request Block" on page 6-ID02-19 for block diagrams.

Service-Specific Input

Size	Offset	Description
DWord	10H	Physical pointer to SCB
Word	14H	Reserved
DWord	16H	Logical pointer to SCB chain header
Word	1CH	Reserved
Word	26H	Reserved
Word	2CH	Reserved
Byte	2EH	Flags Bits 7 to 1 - Reserved (set to 0) Bit 0 - SCB length = 0 - Normal-length SCB = 1 - Long SCB

Service-Specific Output

Size	Offset	Description
DWord	1EH	Logical pointer to chain header of last SCB processed; check the Status field (offset hex 32) for validity
DWord	28H	Time to wait before resuming request, in microseconds
Word	2FH	Soft-error indicator = 00H - Soft error did not occur ≠ 00H - Soft error occurred
Byte	32H	Status Bits 7 to 1 - Reserved Bit 0 = 1 - Pointer at offset hex 1E is valid

13H—Reserved

14H—Deallocate

- This function removes the association between the physical devices and the logical ID.
- All devices that are assigned to the logical ID are released.
- After a logical ID is deallocated:
  - The logical ID cannot be used.
  - The default interrupt handler for that logical ID returns hex 0005 (Not My Interrupt, Stage on Interrupt).
- This function is intended to be used in conjunction with the Allocate SCSI Peripheral Device function (hex 15 in "Device ID 18—SCSI Peripheral Type"). Access to the device is gained through a logical ID by deallocating the disk logical ID. There is an individual SCSI-peripheral-type logical ID for each unit.

- On return, the SCSI Disk Number field contains a value that can be combined with a unit number and used as input for the Allocate function. This allows the controlling program to request the same device (within its class) that it has just released.

  For example, if the disk logical ID has two units, the Allocate SCSI Peripheral Device function must be called twice. The first call is for the first unit; use the value that is returned in the SCSI Disk Number field. The second call is for the second unit; add 1 to the value that is returned in the SCSI Disk Number field.

- The Return Logical ID Parameters function (hex 01), bit 4 of the Logical ID Flags field indicates whether SCSI ABIOS is supported. The Read Device Parameters function (hex 03), bit 14 of the Device Control Flags field indicates whether the device is a SCSI device.

- If any of the devices that are assigned to this logical ID are busy, the Return Code field is set to hex 8000 (Device Busy, Request Refused).

- The possible values of the Return Code field are hex 0000, 8000, and 8003.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Word	12H	SCSI disk number - Can be used in the Allocate SCSI Peripheral Device function to identify which disk within this class is to be requested (offset hex 12 on input)

18H—Set Physical Pointer to Device Block

- A physical pointer to the device block is stored in each SCSI fixed disk device block. This function should be called if the operating system moves the device block after ABIOS is initialized. This function updates the stored physical pointer.

- Each device block must be in physically-contiguous memory.

- The physical pointer is range checked to ensure that the device block will fit in memory.

- The possible values of the Return Code field are hex 0000 and C006.

Service-Specific Input

Size	Offset	Description
Word	18H	Reserved
DWord	1AH	Physical pointer to device block
Word	1EH	Reserved

Service-Specific Output

Size	Offset	Description
None		

19H—Return Pointers to Device Block

- A physical pointer to the device block is stored in each SCSI fixed disk device block. This function returns the logical pointer to the device block (as passed in the ABIOS stack frame on this call) and the current physical pointer that is stored in the device block.
- The information that is returned in this function is to be used by operating systems that relocate the device blocks after ABIOS is initialized. The physical pointer might or might not be valid. If the device block is moved after ABIOS is initialized, the operating system must update the physical pointer to enable ABIOS to transfer command blocks to the hardware.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	12H	Logical pointer to device block
DWord	1AH	Physical pointer to device block
Word	20H	Device-block length

1AH—Return SCSI-Specific Parameters

- This function returns information about the SCSI fixed disk.
- Using the returned information to program the hardware directly is not recommended.
- To protect the IML portion of the fixed disk, use the Read Device Parameters function (hex 03) to determine the maximum available relative block address.
- The possible values of the Return Code field are hex 0000 and 8003.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved

Service-Specific Output

Size	Offset	Description
Byte	12H	Physical unit number
Byte	13H	Logical unit number
Byte	14H	Logical device number
Byte	15H	Adapter index (0 based)
Word	16H	Base port address
Word	18H	Reserved
Word	1AH	Reserved

Return Codes

Return codes are returned at offset hex 0C.

Value	Description
0000H	Operation Successfully Completed
0001H	Stage on Interrupt
0002H	Stage on Time
0005H	Not My Interrupt, Stage on Interrupt
8000H	Device Busy, Request Refused
8001H	Device Not Powered-On
8002H	Device Block Not Properly Initialized
8003H	Device Not Allocated
800FH	DMA Arbitration Level Out of Range
8100H	Retryable Device Busy, Request Refused
9001H	Bad Command
9002H	Address Mark Not Found
9003H	Write-Protect Error
9004H	Record Not Found
9005H	Reset Failed
9006H	Media Changed
9007H	Controller Parameter Activity Failed
9008H	DMA Failed
900AH	Defective Sector
900BH	Bad Track
900DH	Format Error
900EH	CAM Detected during Read or Verify
9010H	Uncorrectable ECC or CRC Error
9014H	Device Failed
9015H	Bus Fault
9020H	Bad Controller
9021H	Equipment Check
9040H	Bad Seek
9080H	Device Did Not Respond
90AAH	Drive Not Ready
90BBH	Undefined Error
90CCH	Write Fault
90E0H	Status Error
90FFH	Incomplete Sense Operation
9101H	Bad Command
9102H	Address Mark Not Found
9103H	Write-Protect Error
9104H	Record Not Found
9105H	Reset Failed
9106H	Media Changed
9107H	Controller Parameter Activity Failed
9108H	DMA Failed
9114H	Device Failed
9115H	Bus Fault
9120H	Bad Controller
9121H	Equipment Check

Figure 6-8 (Part 1 of 3). Fixed Disk Return Codes

Value	Description
9140H	Bad Seek
9180H	Device Did Not Respond
91AAH	Drive Not Ready
91BBH	Undefined Error
91CCH	Write Fault
91E0H	Status Error
91FFH	Incomplete Sense Operation
A000H	Time-Out Occurred, No Other Error
A001H	Bad Command
A002H	Address Mark Not Found
A004H	Record Not Found
A005H	Reset Failed
A007H	Parameter Activity Failed
A00AH	Defective Sector
A00BH	Bad Track
A00DH	Invalid Sector on Format
A00EH	CAM Detected during Read or Verify
A010H	Uncorrectable ECC or CRC Error
A011H	ECC Corrected Data Error
A020H	Bad Controller
A021H	Equipment Check
A040H	Bad Seek
A080H	Device Did Not Respond
A0AAH	Drive Not Ready
A0BBH	Undefined Error
A0CCH	Write Fault
A0FFH	Incomplete Sense Operation
A100H	Time-Out Occurred, No Other Error
A105H	Reset Failed
A107H	Controller Parameter Activity Failed
A120H	Bad Controller
A121H	Equipment Check
A140H	Bad Seek
A180H	Device Did Not Respond
A1AAH	Drive Not Ready
A1BBH	Undefined Error
A1CCH	Write Fault
A1FFH	Incomplete Sense Operation
B001H	Bad Command
B020H	Bad Controller
B021H	Equipment Check
B080H	Device Did Not Respond
B0BBH	Undefined Error
B0FFH	Sense Failed
B101H	Bad Command
B120H	Bad Controller
B121H	Equipment Check
B180H	Device Did Not Respond

Figure 6-8 (Part 2 of 3). Fixed Disk Return Codes

Value	Description
B1BBH	Undefined Error
B1FFH	Sense Failed
C000H	Invalid Logical ID (ABIOS transfer convention only)
C001H	Invalid Function
C003H	Invalid Unit Number
C004H	Invalid Request-Block Length
C005H	Invalid Count Value
C006H	Range Exceeded
C007H	Invalid Disk Parameter

Figure 6-8 (Part 3 of 3). Fixed Disk Return Codes

Programming Considerations

- In ABIOS, the disk interface requires the use of the DMA ABIOS interface; therefore, if the disk routines are initialized and used, the DMA ABIOS routines must be initialized.
- The Read Device Parameters (hex 03) returns the number of retries for any one operation when an error occurs.
- When an error occurs, ABIOS resets the disk system, if necessary.
- For the Read, Write, and Write/Verify functions, the output parameter at offset hex 2C represents the number of blocks that were transferred, as determined by the hardware. This value is supplied when the request ends in an error before the data transfer is completed. If no error is reported, this value equals the number of blocks that were requested to be transferred. This value is valid only when the request is completed (successfully or unsuccessfully).
- When error-recovery procedures are successful and reported, fixed disk routines attempt to determine the nature of the recovery procedures that were performed. Fixed disk routines set the Soft Error field in the transfer SCB request block with the recovered error code.
- Relative block addresses begin ordering with the first disk block that is assigned a value of 0. For hardware devices that do not support relative block addresses, the equivalent is cylinder 0, head 0, and sector 1. In the following formulas, "sectors per track," "sector ID," "heads," and "cylinders" refer to physical (1 based) entities. "Cylinder" and "head" refer to ID values as they are actually sent to the controller (0 based). Fixed disk ABIOS returns physical values for the number of sectors per track, the number of heads, and the number of cylinders in the Read Device

Parameters function (hex 03). These values should be used for relative-block-address calculations. ABIOS uses the following formulas to break down the relative block address (RBA):

```
Sector ID = (RBA MOD Sectors per track) + 1

Head = (RBA/Sectors per track) × MOD heads

Cylinder = (RBA/Sectors per track)/Heads
```

The RBA is calculated as follows:

```
RBA = (Sectors per track × Heads × Cylinder) +
      (Sectors per track × Head) + (Sector ID - 1)
```

The number of RBAs is calculated as follows (this is the value that is returned by the Read Device Parameters function):

```
RBAs = Cylinders × Heads × Sectors per track
```

The maximum allowable RBA is calculated as follows:

```
Maximum RBA = (Cylinders × Heads × Sectors per track) - 1
```

- When fixed disk ABIOS and fixed disk BIOS requests are issued, the following rules must be followed:
  - Do not attempt an ABIOS call while there is an outstanding BIOS call.
  - Do not attempt a BIOS call while there is an outstanding ABIOS call.
  - The Reset/Initialize function (hex 05) must be issued after ABIOS initialization has been completed.

Device Control Flags

The following flags are returned at offset hex 14 of the Read Device Parameters function (hex 03).

SCB transfer support This bit indicates whether ABIOS supports the Transfer SCB function (hex 12).

SCSI device This bit indicates whether the attached device is a SCSI device. It can also be used to test for the validity of the LUN field. This interface supports SCSI peripheral type-0 devices with nonremovable media and 512-byte block states.

Power off This bit reflects the power state of the device at initialization. When this bit is set to 1, the device is powered-off. When this bit is set to 0, the device is powered-on.

Parameter validity This bit indicates the validity of the returned values in the Sector, Head, Block Size, Cylinder, and Maximum RBA fields. When this bit is set to 1, the parameters are not valid. When this bit is set to 0, the parameters are valid. In most cases, when the parameters are not valid, it is because the disk drive was powered-off during ABIOS initialization.

Device Block

Subsystem control blocks (SCB) and termination status blocks (TSB) require a physical pointer for the adapter. Therefore, the physical pointer to the device block (as calculated during ABIOS initialization) is kept in the device block and used when the SCB or TSB is required. This requirement places a restriction on relocating the disk device block. It cannot be relocated after ABIOS is initialized.

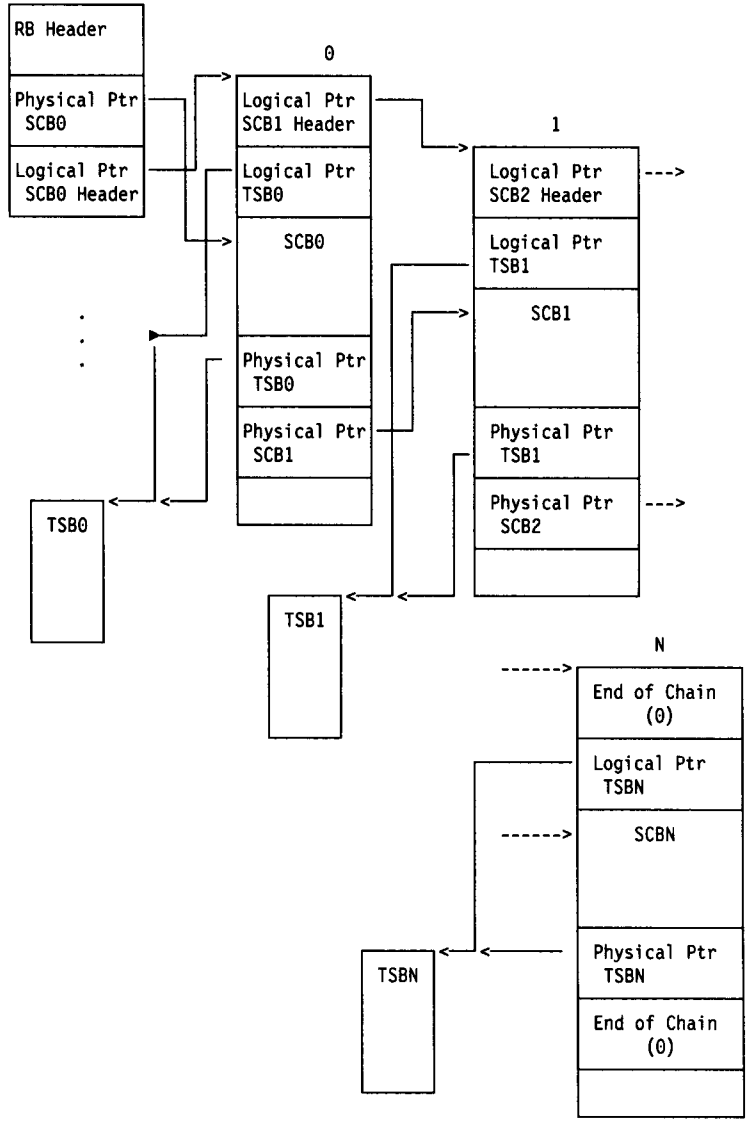
Transfer SCB Request Block

Size	Offset	Description
DWord	10H	Physical pointer to SCB
Word	14H	Reserved
DWord	16H	Logical pointer to SCB chain header
Word	1CH	Reserved
DWord	1EH	Logical pointer to last SCB that was processed
Word	26H	Reserved
DWord	28H	Time to wait before resuming request (stage on time)
Word	2CH	Reserved
Byte	2EH	Flags
Word	2FH	Soft error
Byte	32H	Status (byte)

Chain Header

Size	Offset	Description
Word	00H	Reserved
DWord	02H	Logical pointer to next SCB chain header
Word	06H	Reserved
Word	08H	Reserved
DWord	0AH	Logical pointer to TSB
Word	0EH	Reserved
	10H	Start of SCB

Chain Example



Device ID 03H—Video

Functions

The following are the video functions. The Default Interrupt Handler function (hex 00) and the Return Logical ID Parameters function (hex 01) are described in "Request Block" in the "Transfer Conventions" section.

Note: All reserved input fields must be set to 0.

00H—Default Interrupt Handler

01H—Return Logical ID Parameters

02H—Reserved

03H—Read Device Parameters

- This function returns parameters that indicate the current video state.
- The Character Block Specifier field returns the active character-generator blocks. The Character Block Select A field specifies the block that is used to generate alphanumeric characters when bit 3 of the character-attribute byte is set to 1. The Character Block Select B field specifies the block that is used to generate alphanumeric characters when bit 3 of the character-attribute byte is set to 0. When the value in the Character Block Select A field is equal to the value in the Character Block Select B field, character selection is disabled, and bit 3 of the character-attribute byte determines the foreground-intensity state (1 = On, 0 = Off).
- The Save/Restore Header Size field, Hardware State Size field, Device Block State Size field, and DAC State Size field are used in calculating the size of the save buffer for the Save Environment function (hex 0C). Refer to the Save Environment function (hex 0C) on page 6-ID03-5 for more information.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	28H	Reserved

Service-Specific Output

Size	Offset	Description
Byte	1CH	Number of scan lines on the screen = 00H - 200 scan lines = 01H - 350 scan lines = 02H - 400 scan lines = 03H - 480 scan lines = 04H to 0FFH - Reserved
Word	1EH	Video mode setting (see Figure 6-10 on page 6-ID03-14)
Word	20H	Type of display attached Bits 15 to 1 - Reserved Bit 0 - Color or monochrome = 0 - Color display = 1 - Monochrome display
Word	22H	Character height (bytes per character)
Word	24H	Character-block specifier Bits 15 to 12 - Reserved Bits 11 to 8 - Character block select A Bits 7 to 4 - Reserved Bits 3 to 0 - Character block select B
Word	2AH	Size of data buffer that is required for the Return ROM Fonts Information function (hex 0B)
Word	2EH	Size of the save/restore buffer header, in bytes
Word	30H	Size of the save/restore hardware state, in bytes
Word	32H	Size of the save/restore device-block state, in bytes
Word	34H	Size of the save/restore digital-to-analog converter (DAC) state, in bytes

04H—Set Device Parameters (Reserved)

05H—Reset/Initialize

- This function initializes the video controller to the requested mode (see Figure 6-10 on page 6-ID03-14).
- The Character Blocks to Be Loaded field indicates which character blocks will be loaded with the default ROM character font for the specified mode and number of scan lines. This parameter is required only when an alphanumeric mode (hex 0, 1, 2, 3, 7, or 14) is being set.
- The Number of Scan Lines field and the Character Block Specifier field are specified only when an alphanumeric mode (hex 0, 1, 2, 3, 7, or 14) is being set.
- In the Character Block Specifier field, the Character Block Select A field specifies the block that is used to generate alphanumeric characters when bit 3 of the character-attribute byte is set to 1. The Character Block Select B field specifies the block that is used to generate alphanumeric characters when bit 3 of the character-attribute byte is set to 0. When the value in the Character Block Select A field is equal to the value in the

Character Block Select B field, character selection is disabled, and bit 3 of the character-attribute byte determines the foreground-intensity state (1 = On, 0 = Off).

- The summing bit of the Device Control Flags field is required only when a color display is attached. Summing is performed automatically for monochrome displays.
- When a monochrome display is used in a color mode, the colors are displayed as shades of gray. Of 64 gray shades, 16 are available in all modes except mode hex 13. In mode hex 13, all 64 gray shades are available.
- Modes hex 0, 2, and 4 are identical to modes hex 1, 3, and 5, respectively.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	1AH	Device control flags Bits 15 to 3 - Reserved Bit 2 - Summing = 0 - Summing disabled = 1 - Summing enabled Bit 1 - Initialize digital-to-analog converter (DAC) to default = 0 - Do not initialize DAC to default = 1 - Initialize DAC to default Bit 0 - Regenerative buffer flag = 0 - Do not clear buffer = 1 - Clear buffer
Byte	1CH	Number of scan lines = 00H - 200 scan lines (modes 0, 1, 2, 3, and 14) = 01H - 350 scan lines (modes 0, 1, 2, 3, 7, and 14) = 02H - 400 scan lines (modes 0, 1, 2, 3, 7, and 14) = 03H to FFH - Reserved
Word	1EH	Video mode to be set (see Figure 6-10 on page 6-ID03-14)
Word	24H	Character block specifier Bits 15 to 12 - Reserved Bits 11 to 8 - Character block select A Bits 7 to 4 - Reserved Bits 3 to 0 - Character block select B
Word	26H	Character blocks to be loaded with default ROM font Bit *n* - Block *n* flag = 0 - Do not update font = 1 - Update font
Word	28H	Reserved

Service-Specific Output

Size	Offset	Description
None		

06H—Enable (Reserved)

07H—Disable (Reserved)

08H—Read (Reserved)

09H—Write (Reserved)

0AH—Additional Data Transfer Function (Reserved)

0BH—Return ROM Fonts Information

- This function returns the following information about each of the ROM fonts:
  - The pointer to the ROM font
  - The character size (row and column)
  - Whether it is a total font or a partial font
  - If it is a partial font, which font it relates to.
- There are 12 bytes of information for each ROM font. They are stored sequentially in the specified data area.
- Each ROM-font entry has the following format:

```
Word  - Reserved
DWord - Pointer to ROM font
Word  - Reserved
Byte  - Character size (number of columns)
Byte  - Character size (number of rows)
Byte  - Total-/partial-font indicator
        = 00H - Total font
        = 01H - Partial font
        = 02H to FFH - Reserved
Byte  - Related font
        If this is a partial font, this byte contains a
        number to indicate which font this font goes with.
        The font number is based on the place a particular
        font occupies in the ROM-font entries.
```

- Before this function is used, the Read Device Parameters function (hex 03) must be issued to determine the size of the buffer that is required to save the ROM-fonts information.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Pointer to buffer to store ROM-fonts information
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
None		

0CH—Save Environment

- This function stores the caller's requested video states in the specified buffer.
- The video environment consists of the following states:
  - Hardware state
  - Device block state
  - Digital-to-analog converter (DAC) state.
- To calculate the size of the save buffer that is required, the Read Device Parameters function (hex 03) must be issued. It gives the individual sizes of the possible states to be saved and the size of the save/restore header. Then:

  Save-buffer size = (A + B + C + D)

  where:

  A = Size of the save/restore header
  B = Environment (bit 0) × (size of hardware state)
  C = Environment (bit 1) × (size of device block state)
  D = Environment (bit 2) × (size of DAC state)

- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Pointer to environment save area
Word	16H	Reserved
Word	2CH	Video-environment states to be saved Bits 15 to 3 - Reserved (set to 0) Bit 2 - DAC state = 1 - Save state Bit 1 - Device block state = 1 - Save state Bit 0 - Hardware state = 1 - Save state

Service-Specific Output

Size	Offset	Description
None		

0DH—Restore Environment

- This function restores the video environment from the specified buffer location. Refer to the Save Environment function (hex 0C) for more information about the contents and structure of the video environment.
- Restoring a state that was not previously saved can cause unpredictable results.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Pointer to environment restore area
Word	16H	Reserved
Word	1AH	Device control flag Bits 15 to 1 - Reserved Bit 0 - Regenerative-buffer flag = 0 - Do not clear buffer = 1 - Clear buffer
Word	2CH	Video-environment states to be restored Bits 15 to 3 - Reserved (set to 0) Bit 2 - DAC state = 1 - Restore state Bit 1 - Device block state = 1 - Restore state Bit 0 - Hardware state = 1 - Restore state

Service-Specific Output

Size	Offset	Description
None		

0EH—Select Character-Generator Block

- This function selects up to two character-generator blocks.
- In the Character Block Specifier field, the Character Block Select A field specifies the block that is used to generate alphanumeric characters when bit 3 of the character-attribute byte is set to 1. The Character Block Select B field specifies the block that is used to generate alphanumeric characters when bit 3 of the character-attribute byte is set to 0. When the value in the Character Block Select A field is equal to the value in the

Character Block Select B field, character selection is disabled, and bit 3 of the character-attribute byte determines the foreground-intensity state (1 = On, 0 = Off).

- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved
Word	24H	Character-block specifier Bits 15 to 12 - Reserved Bits 11 to 8 - Character block select A Bits 7 to 4 - Reserved Bits 3 to 0 - Character block select B

Service-Specific Output

Size	Offset	Description
None		

0FH—Alphanumeric Load

- This function loads the requested character generator, or part of it, to the specified character blocks.
- This function does not update the hardware registers. Refer to the Enhanced Alphanumeric Load function (hex 10) if hardware updating is required.
- When any of the ROM character generators is being loaded (the Character Generator Type field is set to 1, 2, or 3), the full set of characters (hex 100) is loaded. Therefore, the only parameters that are required to invoke this function are the Character Generator Type field and the Character Block Specifier field.
- When a user font is being loaded (the Character Generator Type field is set to 0), all parameters are required.
- When a user font is being loaded, if the Count of Characters field is set to 0, no character is loaded, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- When a user font is being loaded, the sum of the values in the Count of Characters field and the Character Offset field must not exceed the maximum valid number of characters in a set (hex 100). If the sum does exceed the maximum valid number of characters, the Return Code field is set to hex C005 (Invalid Video Parameter).
- The possible values of the Return Code field are hex 0000 and C005.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Pointer to user font
Word	16H	Reserved
Word	18H	Count of characters = 001H to 100H - Valid count of characters
Byte	1DH	Character-generator type: = 00H - User's alphanumeric font = 01H - 8x8 alphanumeric ROM font = 02H - 8x14 alphanumeric ROM font = 03H - 8x16 alphanumeric ROM font = 04H to FFH - Reserved
Word	22H	Character height (bytes per character)
Word	24H	Character block to be loaded = 0000H to 0007H - Valid values of character blocks to be loaded = 0008H to FFFFH - Reserved
Word	28H	Character offset into the table

Service-Specific Output

Size	Offset	Description
None		

10H—Enhanced Alphanumeric Load

- This function loads the requested character generator, or part of it, to the specified character block and updates the hardware registers.
- When any of the ROM character generators is being loaded (the Character Generator Type field is set to 1, 2, or 3), the full set of characters (hex 100) is loaded. Therefore, the only parameters that are required to invoke this function are the Character Generator Type field and the Character Blocks to Be Loaded field.
- When a user font is being loaded (the Character Generator Type field is set to 0), all parameters are required.
- When a user font is being loaded, if the Count of Characters field is set to 0, no character is loaded, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- When a user font is being loaded, the sum of the values of the Count of Characters field and the Character Offset field must not exceed the maximum valid number of characters in a set (hex 100). If the sum does exceed the maximum valid number of characters, the Return Code field is set to hex C005 (Invalid Video Parameter).

- The possible values of the Return Code field are hex 0000 and C005.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Pointer to user font
Word	16H	Reserved
Word	18H	Count of characters = 001H to 100H - Valid count of characters
Byte	1DH	Character-generator type = 00H - User's alphanumeric font = 01H - 8x8 alphanumeric ROM font = 02H - 8x14 alphanumeric ROM font = 03H - 8x16 alphanumeric ROM font = 04H to FFH - Reserved
Word	22H	Character height (bytes per character)
Word	24H	Character block to be loaded = 0000H to 0007H - Valid values of character blocks to be loaded = 0008H to FFFFH - Reserved
Word	28H	Character offset into the table

Service-Specific Output

Size	Offset	Description
None		

11H—Read Palette Register

- This function reads a palette register.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved
Word	32H	Palette register to be read = 0000H to 000FH - Valid values of palette register to be read = 0010H to FFFFH - Reserved

Service-Specific Output

Size	Offset	Description
Word	34H	Palette value that was read

12H—Write Palette Register

- This function writes a value to a palette register.
- Execution of this function is not supported in mode hex 13. The hardware requires that the values in these registers not be changed after they are set by the Reset/Initialize function (hex 05). Changing the values in these registers can cause unpredictable results.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved
Word	32H	Palette register to be written = 0000H to 000FH - Valid values of palette register to be written = 0010H to FFFFH - Reserved
Word	34H	Palette value to be loaded = 0000H to 003FH - Valid palette values = 0040H to FFFFH - Reserved

Service-Specific Output

Size	Offset	Description
None		

13H—Read Color Register

- This function reads the red, green, and blue values of a color register from the video digital-to-analog converter.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved
Word	2AH	Color register to be read = 0000H to 00FFH - Valid values of color register to be read = 0100H to FFFFH - Reserved

Service-Specific Output

Size	Offset	Description
Word	2CH	Red value that was read
Word	2EH	Green value that was read
Word	30H	Blue value that was read

14H—Write Color Register

- This function loads a digital-to-analog converter color register with the specified red, green, and blue values.
- In the Device Control Flags field, the summing bit is disregarded when a monochrome display is attached. Summing always occurs with a monochrome display that is operating in a color mode.
- The value of the Return Code field is hex 0000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved
Word	1AH	Device control flags Bits 15 to 3 - Reserved Bit 2 - Summing = 0 - Summing disabled = 1 - Summing enabled Bits 1, 0 - Reserved
Word	2AH	Color register to be written = 0000H to 00FFH - Valid values of color registers to be written = 0100H to FFFFH - Reserved
Word	2CH	Red value to be written = 0000H to 003FH - Valid red values to be written = 0040H to FFFFH - Reserved
Word	2EH	Green value to be written = 0000H to 003FH - Valid green values to be written = 0040H to FFFFH - Reserved
Word	30H	Blue value to be written = 0000H to 003FH - Valid blue values to be written = 0040H to FFFFH - Reserved

Service-Specific Output

Size	Offset	Description
None		

15H—Read Block of Color Registers

- This function reads a block of digital-to-analog converter color registers into the specified save area, beginning at the specified color register.
- The format of the data that is returned is "red value, green value, blue value, red value, green value, blue value, . . . , red value, green value, blue value."
- The range for the red, green, and blue values is from hex 00 to hex 3F.

- If the Number of Color Registers to Be Read field is set to 0, no action is performed, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- If the sum of the values of the First Color Register to Be Read field and the Number of Color Registers to Be Read field is greater than the maximum valid number of color registers, no action is performed, and the Return Code field is set to hex C005 (Invalid Video Parameter).
- The possible values of the Return Code field are hex 0000 and C005.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Pointer to read save area
Word	16H	Reserved
Word	18H	Number of color registers to be read
Word	2AH	First color register to be read = 0000H to 00FFH - Valid values of first color register to be read = 0100H to FFFFH - Reserved

Service-Specific Output

Size	Offset	Description
None		

16H—Write Block of Color Registers

- This function loads a block of digital-to-analog converter color registers with the requested values, beginning with the requested color register.
- The format of the data that is returned is "red value, green value, blue value, red value, green value, blue value, . . . , red value, green value, blue value."
- If the Number of Color Registers to Be Written field is set to 0, no action is performed, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- If the sum of the values of the First Color Register to Be Written field and the Number of Color Registers field is greater than the maximum valid number of color registers, no action is performed, and the Return Code field is set to hex C005 (Invalid Video Parameter).
- In the Device Control Flags field, the summing bit is disregarded when a monochrome display is attached. Summing always

occurs with a monochrome display that is operating in a color mode.

- The possible values of the Return Code field are hex 0000 and C005.

Service-Specific Input

Size	Offset	Description
Word	10H	Reserved
DWord	12H	Pointer to write save area
Word	16H	Reserved
Word	18H	Number of color registers to be written
Word	1AH	Device control flags Bits 15 to 3 - Reserved Bit 2 - Summing = 0 - Summing disabled = 1 - Summing enabled Bits 1, 0 - Reserved
Word	2AH	First color register to be written = 0000H to 00FFH - Valid value of first color register to be written = 0100H to FFFFH - Reserved

Service-Specific Output

Size	Offset	Description
None		

Return Codes

Return codes are returned at offset hex 0C.

Value	Description
0000H	Operation Successfully Completed
C000H	Invalid Logical ID (ABIOS transfer convention only)
C001H	Invalid Function
C003H	Invalid Unit Number
C004H	Invalid Request-Block Length
C005H	Invalid Video Parameter

Figure 6-9. Video Return Codes

Video Modes

The following table shows the supported video modes.

Mode (Hex)	Type	Maximum Colors	A/N Format	Buffer Start	Box Size	Maximum Pages	Display Size
00	A/N	16	40x25	B8000H	8x8	8	320x200
00	A/N	16	40x25	B8000H	8x14	8	320x350
00	A/N	16	40x25	B8000H	8x16	8	320x400
00	A/N	16	40x25	B8000H	9x16	8	360x400
01	A/N	16	40x25	B8000H	8x8	8	320x200
01	A/N	16	40x25	B8000H	8x14	8	320x350
01	A/N	16	40x25	B8000H	8x16	8	320x400
01	A/N	16	40x25	B8000H	9x16	8	360x400
02	A/N	16	80x25	B8000H	8x8	4	640x200
02	A/N	16	80x25	B8000H	8x8	8	640x200
02	A/N	16	80x25	B8000H	8x14	8	640x350
02	A/N	16	80x25	B8000H	8x16	8	640x400
02	A/N	16	80x25	B8000H	9x16	8	720x400
03	A/N	16	80x25	B8000H	8x8	4	640x200
03	A/N	16	80x25	B8000H	8x8	8	640x200
03	A/N	16	80x25	B8000H	8x14	8	640x350
03	A/N	16	80x25	B8000H	8x16	8	640x400
03	A/N	16	80x25	B8000H	9x16	8	720x400
04	APA	4	40x25	B8000H	8x8	1	320x200
05	APA	4	40x25	B8000H	8x8	1	320x200
06	APA	2	80x25	B8000H	8x8	1	640x200
07	A/N	Monochrome	80x25	B0000H	9x14	1	720x350
07	A/N	Monochrome	80x25	B0000H	9x14	8	720x350
07	A/N	Monochrome	80x25	B0000H	9x16	8	720x400
07	A/N	Monochrome	80x25	B0000H	8x8	4	640x200
08	APA	16	20x25	B0000H	8x8	1	160x200
09	APA	16	40x25	B0000H	8x8	1	320x200

Figure 6-10 (Part 1 of 2). Video Modes

Mode (Hex)	Type	Maximum Colors	A/N Format	Buffer Start	Box Size	Maximum Pages	Display Size
0A	APA	4	80x25	B0000H	8x8	1	640x200
0B	Reserved						
0C	Reserved						
0D	APA	16	40x25	A0000H	8x8	8	320x200
0E	APA	16	80x25	A0000H	8x8	4	640x200
0F	APA	Monochrome	80x25	A0000H	8x14	2	640x350
10	APA	16	80x25	A0000H	8x14	2	640x350
11	APA	2	80x30	A0000H	8x16	1	640x480
12	APA	16	80x30	A0000H	8x16	1	640x480
13	APA	256	40x25	A0000H	8x8	1	320x200
14	A/N	16	132x25	B8000H	8x8	4	1056x200
14	A/N	16	132x25	B8000H	8x14	4	1056x350
14	A/N	16	132x25	B8000H	8x16	4	1056x400

APA = All points addressable (graphics)
A/N = Alphanumeric (text)

Figure 6-10 (Part 2 of 2). Video Modes

Notes:

Device ID 04H—Keyboard

Functions

The following are the keyboard functions. The Default Interrupt Handler function (hex 00) and the Return Logical ID Parameters function (hex 01) are described in "Request Block" in the "Transfer Conventions" section.

Note: All reserved input fields must be set to 0.

00H—Default Interrupt Handler

01H—Return Logical ID Parameters

02H—Reserved

03H—Read Device Parameters

- This function returns the keyboard identification values.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 0005, 8000, 8003, 9000, 9002, 9003, 9004, 9100, 9102, 9103, 9104, B001, and B101.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds
Byte	14H	Keyboard ID byte 1
Byte	15H	Keyboard ID byte 2

04H—Set Device Parameters (Reserved)

05H—Reset/Initialize Keyboard

- This function resets the keyboard and turns off the Num Lock, Caps Lock, and Scroll Lock indicator lights.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 0005, 8000, 8003, 9000, 9001, 9002, 9100, 9101, 9102, B001, and B101.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds

06H—Enable

- This function enables the keyboard so that keyboard data can be passed to the system.
- The possible values of the Return Code field are hex 0000, 0002, 8000, 8003, 9000, and 9100.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds

07H—Disable

- This function disables the keyboard by inhibiting the flow of keyboard data to the system.
- The possible values of the Return Code field are equal to hex 0000, 0002, 8000, 8003, 9000, and 9100.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds

08H—Continuous Read

- This function returns keyboard scan codes. It must be called soon after ABIOS initialization to allow for the processing of keystroke interrupts.
- After this function has been started, it is a continuous multistaged request.
- At interrupt time, if a scan code is available, the Keyboard Interrupt routine returns hex 0009 (Attention, Stage on Interrupt) in the Return Code field. This return code indicates that there is a valid scan code in the Keyboard Raw Scan Code field.
- The possible values of the Return Code field are hex 0001, 0005, 0009, and 8000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Byte	14H	Keyboard raw scan code

09H—Write (Reserved)

0AH—Additional Data Transfer (Reserved)

0BH—Read Keyboard Indicators

- This function returns the current state of the keyboard Num Lock, Caps Lock, and Scroll Lock indicator lights.
- The possible values of the Return Code field are hex 0000 and 8000.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
Byte	14H	Keyboard-indicator data Bits 7 to 3 - Reserved Bit 2 - Caps Lock = 0 - Off = 1 - On Bit 1 - Num Lock = 0 - Off = 1 - On Bit 0 - Scroll Lock = 0 - Off = 1 - On

0CH—Write Keyboard Indicators

- This function programs the state of the keyboard Num Lock, Caps Lock, and Scroll Lock indicator lights.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 0005, 8000, 8003, 9000, 9002, 9100, 9102, B001, and B101.

Service-Specific Input

Size	Offset	Description
Byte	14H	Keyboard indicator data Bits 7 to 3 - Reserved (must be set to 0) Bit 2 - Caps Lock = 0 - Off = 1 - On Bit 1 - Num Lock = 0 - Off = 1 - On Bit 0 - Scroll Lock = 0 - Off = 1 - On
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds

0DH—Set Typematic Rate and Delay

- This function changes the current setting of the typematic rate and delay for the keyboard.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 0005, 8000, 8003, 9000, 9002, 9100, 9102, B001, and B101.

Service-Specific Input

Size	Offset	Description
Byte	14H	Rate value Bits 7 to 5 - Reserved (must be set to 0) Bits 4 to 0 - Rate value, in characters per second (values are in binary) = 00000 - 30.0 = 10000 - 7.5 = 00001 - 26.7 = 10001 - 6.7 = 00010 - 24.0 = 10010 - 6.0 = 00011 - 21.8 = 10011 - 5.5 = 00100 - 20.0 = 10100 - 5.0 = 00101 - 18.5 = 10101 - 4.6 = 00110 - 17.1 = 10110 - 4.3 = 00111 - 16.0 = 10111 - 4.0 = 01000 - 15.0 = 11000 - 3.7 = 01001 - 13.3 = 11001 - 3.3 = 01010 - 12.0 = 11010 - 3.0 = 01011 - 10.9 = 11011 - 2.7 = 01100 - 10.0 = 11100 - 2.5 = 01101 - 9.2 = 11101 - 2.3 = 01110 - 8.6 = 11110 - 2.1 = 01111 - 8.0 = 11111 - 2.0
Byte	15H	Delay value Bits 7 to 2 - Reserved (must be set to 0) Bits 1, 0 - Delay value, in milliseconds (values are in binary) = 00 - 250 = 01 - 500 = 10 - 750 = 11 - 1000
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds

0EH—Read Keyboard Mode

- This function returns the current keyboard scan-code mode.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 0005, 8000, 8003, 9000, 9002, 9003, 9004, 9006, 9100, 9102, 9103, 9104, 9106, B001, and B101.

Service-Specific Input

Size	Offset	Description
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds
Byte	14H	Current keyboard scan-code mode = 00H - Reserved = 01H - Scan code set 1 = 02H - Scan code set 2 = 03H - Scan code set 3 = 04H to FFH - Reserved

0FH—Set Keyboard Mode

- This function changes the current keyboard scan-code mode.
- The possible values of the Return Code field are hex 0000, 0001, 0002, 0005, 8000, 8003, 9000, 9002, 9100, 9102, B001, and B101.

Service-Specific Input

Size	Offset	Description
Byte	14H	Keyboard scan-code mode to be set = 00H - Reserved = 01H - Scan code set 1 = 02H - Scan code set 2 = 03H - Scan code set 3 = 04H to FFH - Reserved
Word	16H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds

10H—Write Keyboard-Controller Data String

- This function sends the requested data string to the keyboard controller.
- If the Data String Count field is set to 0, no action is performed, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- The possible values of the Return Code field are hex 0000, 0001, 0002, 0005, 8000, 8003, 9000, 9002, 9100, 9102, B001, and B101.

Service-Specific Input

Size	Offset	Description
Word	14H	Reserved
DWord	16H	Pointer to data area
Byte	1CH	Data string count
Word	28H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds

11H—Write Keyboard Data String

- This function sends the requested data string to the keyboard.
- If the Data String Count field is set to 0, no action is performed, and the Return Code field is set to hex 0000 (Operation Successfully Completed).
- The possible values of the Return Code field are hex 0000, 0001, 0002, 0005, 8000, 8003, 9000, 9002, 9100, 9102, B001, and B101.

Service-Specific Input

Size	Offset	Description
Word	14H	Reserved
DWord	16H	Logical pointer to data area
Byte	1CH	Data string count
Word	28H	Reserved

Service-Specific Output

Size	Offset	Description
DWord	10H	Time to wait before resuming request, in microseconds

Return Codes

Return codes are returned at offset hex 0C.

Value	Description
0000H	Operation Successfully Completed
0001H	Incomplete – Stage on Interrupt
0002H	Incomplete – Stage on Time (service specific)
0005H	Incomplete – Not My Interrupt, Stage on Interrupt
0009H	Attention, Stage on Interrupt
8000H	Device Busy – Request Refused
8003H	Security Enabled, Keyboard Inhibited – Request Refused
9000H	Keyboard Controller Perpetually Busy
9001H	Keyboard Failed Reset
9002H	Resend Error
9003H	Keyboard Parity Error
9004H	General Hardware Time-Out
9006H	Undefined Mode Returned by Keyboard
9100H	Keyboard Controller Perpetually Busy
9101H	Keyboard Failed Reset
9102H	Resend Error
9103H	Keyboard Parity Error
9104H	General Hardware Time-Out
B001H	Keyboard Error
B101H	Keyboard Error
C000H	Invalid Logical ID (ABIOS transfer convention only)
C001H	Invalid Function
C003H	Invalid Unit Number
C004H	Invalid Request-Block Length
FFFFH	Return Code Is Not Valid

Figure 6-11. Keyboard Return Codes

Programming Considerations

- Keyboard ABIOS does not attempt any retries. The calling program is responsible for performing retries.
- The Write Keyboard Data String function (hex 11) sends bytes to the keyboard and expects an acknowledgment (ACK) after each byte is sent.
- The Write Keyboard Controller Data String function (hex 10) sends bytes to the keyboard controller; it does not expect a response.
- The Read Keyboard Indicators function (hex 0B) reflects the state of the indicators after either a successful Reset/Initialize Keyboard function (hex 05) or a successful Write Indicators function (hex 0C). If the Write Keyboard Data String function (hex 11) is used to change the indicators, the value that is returned by

the Read Indicators function (hex 0B) might not reflect the true state of the keyboard.

- The Keyboard Time-Out routine does not attempt to reset the keyboard; rather, it sets the Return Code field to hex B001 or hex B101 (Keyboard Error). The calling program is responsible for executing the Keyboard Reset/Initialize function (hex 05).
- If keyboard ABIOS expects an acknowledgment from the keyboard after a command is issued to the keyboard, it does not pass the acknowledgment to the controlling program.

Notes: