Embedded BIOS™ 4.1

The Full-Featured BIOS for Embedded Systems and Consumer Electronics*

BIOS User's Manual with BIOS Interrupt Reference

TABLE OF CONTENTS

Chapter 1

KEY EMBEDDED BIOS CONCEPTS

This chapter presents an architectural overview of EMBEDDED BIOS. OEMs with an understanding of these concepts generally produce BIOSes more efficiently in two ways. First, an appreciation of all the functional issues is an important thing to have before starting a design, so that the design can accommodate those issues. Second, with this material as background, the OEM will have a longer view of the adaptation process. Understanding this material will make your adaptation move more smoothy.

1.1 Architectural Overview

EMBEDDED BIOS is functionally similar to the BIOS in a PC, in many ways. First, the BIOS tests and initializes all of the equipment on the system when power is applied. Once the system has been initialized, it transfers control to an operating system or application. Finally, it provides software services through architected mechanisms that allow the operating system and application to manipulate the hardware; for example, to perform floppy disk I/O, read keystrokes from the keyboard, and display characters on a video display.

Because the BIOS is ultimately responsible for managing the hardware, it must implement policies for initialization and management of the devices. For example, the BIOS's *memory model* determines how much memory will be available to operating systems and applications, and where the memory will be located in the address space.

Similarly, its *interrupt model* determines the policy used to make interrupt assignments of external hardware devices, establish their priorities, and define how operating system and application software will request services from the BIOS.

The BIOS *Power-On Self-Test* (commonly, POST) is responsible for testing and initializing the hardware components in the target such as the DMA controllers, interrupt controllers, programmable timers, and other components so that they work together to provide a viable environment. For example, if dynamic RAM (DRAM) is used in a design, it must be periodically refreshed; this is the responsibility of the BIOS. Using configuration options, the developer directs POST to provide refresh through on-board CPU functions, through chipset functionality, or using more elaborate techniques such as tying an 8254 programmable interval timer to an 8237

DMA controller to cause DMA cycles to perform the refreshing. POST sets up the policies to be used for performing DRAM refresh and many other tasks so that operating systems and applications don't have to do these tasks by themselves.

These and many other architectural issues are described in detail in this chapter.

1.1.1 Memory Model

EMBEDDED BIOS employs a memory model that is compatible with desktop PC standards. Because the BIOS is used primarily in a real-mode environment, it does not define any standards for the use of extended memory beyond 1MB. Instead it is concerned with the layout and usage of memory below 1MB in the address space.

Because Intel-architecture processors can be programmed to respond to a variety of different kinds of addresses (physical, linear, virtual, and real-mode addresses), we will refer to 32-bit physical addresses whenever describing where some object is located in the target machine. When referring to how the object is referenced with actual machine instructions, we will use what is called 16:16 notation for addresses. In this format, addresses contain two parts, each 16 bits in width. The first 16-bit entity is a segment address, and the second 16-bit entity is a byte offset relative to the specified segment. A segment address can be transformed into a physical address by multiplying it by 16 (10h in hexadecimal).

1.1.1.1 The Interrupt Vector Table

At physical location 00000000h in the address space is the real-mode Interrupt Vector Table, or IVT. This table is defined by Intel 80x86 architecture and by other PC standards to be an array of far (16:16) pointers to objects, some being Interrupt Service Routines (ISRs), while other elements are pointers to data structures. This table contains 256 elements and each element is four bytes long, so the table is exactly 1KB in size.

1.1.1.2 The BIOS Data Area

The first address immediately following the IVT is 00000400h. Addressed with the equivalent real-mode segment 0040h, the space following the IVT is called the BIOS Data Area, or BDA. The BDA is used by the BIOS to keep track of how the system is configured; i.e., how many serial and parallel ports exist. It is also used to keep track of the state of the running BIOS, such as the track number over which a floppy disk recording head is positioned. The BDA extends up to but not including physical address 00000500h, so that the first free address to be used by operating systems and application program is 00000500h.

A complete map of the BIOS Data Area is presented in Appendix B, in the actual assembly language source code found in EMBEDDED BIOS. All the fields in the BDA are architected by IBM. Slight modifications to this area have been made by other desktop BIOS vendors since PC clones have matured, to accommodate new BIOS functionality. When these modifications become industry-standard on the desktop, they are incorporated into the EMBEDDED BIOS BDA.

1.1.1.3 Free Low RAM

Starting at physical address 00000500h, or segment 0050h, operating systems and user programs use memory as they see fit. The amount of memory, or size of free low RAM (including the IVT and BDA), is kept in the BIOS Data Area by the BIOS itself, and can be retrieved with a BIOS software service (INT 12h.)

1.1.1.4 The Extended BIOS Data Area

The last 1KB of low memory is reserved by the BIOS for extending the BIOS Data Area without interfering with the well-established user address, 00000500h. During POST, the BIOS determines the amount of low RAM, and reserves the top 1KB of this RAM for itself. When the operating system or user application use the INT 12h BIOS service to determine the amount of low memory, the BIOS actually returns 1KB less than is actually present. In a desktop PC environment, the Extended BIOS Data Area usually ends at physical address 000A0000h to make room for video adapter hardware such as the VGA screen regeneration memory). In designs that do not have VGA hardware at segment A000h, additional memory can be mapped to this address space by the hardware (or possibly by the chipset), so that the BIOS can provide access to a larger amount of low memory.

1.1.1.5 Expanded Memory

In the 1980's a standard emerged for add-on memory cards that provided 64KB pages of memory within the memory range 000A0000h - 000E0000h called expanded memory. Several application programs, such as Lotus 1-2-3 and Windows for example, took advantage of this memory to store program data while they were running. This standard was primarily for application programs, but operating systems evolved to manage this memory. The BIOS, however, never manages this memory by itself (EMBEDDED BIOS does not provide any support for EMS by itself).

1.1.1.6 Video ROM Extensions

Physical address 000C0000h or 000E0000h is inspected by the BIOS during POST for the presence of a possible EGA or VGA ROM BIOS Extension. By checking for a special signature and checksumming the ROM, the BIOS determines if the ROM exists, and if so, it is invoked by the BIOS POST to initialize any video hardware that the core system BIOS is not aware of. For example, the common VGA screens used in desktop PCs are actually not directly supported by the video BIOS on the PC motherboard; instead, the video ROM BIOS Extension on the VGA controller card hooks the BIOS service (INT 10h) so that it can handle video requests instead of the system BIOS.

If a video ROM is not detected by the BIOS, and video services are enabled by the adaptation engineer, then the default video routines in the video module of the BIOS are used to provide video service for monochrome and color graphics adapters.

1.1.1.7 Other ROM Extensions

Additional ROM extensions are detected by POST during system initialization within a special address range of 000C8000h - 000EE000h at 2KB intervals using a special signature pattern and checksum technique. When valid ROM extensions are found, they are called just as video ROM extensions are called, and they perform operations as necessary to support their function. For example, SCSI disk controllers may have ROM BIOS extensions to provide basic disk services

(INT 13h) so that the bootstrap process can actually boot from a SCSI device. Similarly, network interface cards (NICs) may have a remote boot ROM that gets control as a ROM extension so that it can initialize the NIC and request a download of the operating system over a network.

1.1.1.8 The System ROM

The BIOS itself is stored in ROM so that it fits neatly at the end of the 1MB address space. Typically, a 64KB ROM such as a 27C512, or a bulk Flash part such as a 28F010, is used to hold the system BIOS code itself. This code receives control at power-on reset time at physical address 000FFFF0h; this address is equivalent to the 16:16 address F000:FFF0.

On 80386 and above CPUs, the high bits of the physical address are all set, requiring the glue hardware surrounding the CPU to either double-map the 64KB ROM BIOS segment into the top of extended memory, or to disable the high bits so that the CPU really boots from the top of the lower 1MB address space.

Regardless of how the CPU gets control, the system ROM usually occupies 64KB, although the BIOS may be configured to use from 1KB to 64KB of that total file's size with a build option. Naturally, features must be removed from a full-featured 64KB BIOS to allow its size to be reduced.

1.1.1.9 Extended Memory

Just as the BIOS sizes low memory below 1MB during POST, it also determines the amount of RAM above the 1MB address line and keeps this size in CMOS, if available. The amount of usable extended memory is returned through a BIOS software service (INT 15h, function 88h), although the BIOS does not provide any other services for managing this memory beyond simple data copying functionality (INT 15h, function 87h). The management of extended memory is the function of operating system software such as HIMEM.SYS. This driver is available in the Embedded DOS-ROM source tree.

1.1.1.10 CMOS Memory

Actually separate from the memory address space of the processor, an amount of battery-backed CMOS RAM is usually available in AT-compatible systems. In such a compatible configuration, this memory is accessed by reading and writing to I/O ports 70h and 71h.

The BIOS uses this memory to store the equipment configuration and user options associated with the operation of the BIOS, and the integrated BIOS Setup screen system is used to edit the CMOS memory in a running system.

1.1.2 Interrupt Model

In addition to defining the way memory is used in a system, EMBEDDED BIOS has an interrupt model for receiving BIOS service requests via software interrupts, handling CPU traps and faults, processing device hardware interrupts, and managing points in the IVT that point to data structures used by BIOS service modules.

The following table shows the IVT entries used by EMBEDDED BIOS. Note that some interrupts (notably, vectors 08h through 12h) are used by the BIOS although they also be be generated by the CPU in protected mode circumstances.

Vector	Type	Function or Service
00h	CPU	Divide by zero trap
01h	CPU	Single-step trap
02h	CPU	NMI interrupt
03h	CPU	Breakpoint trap (INT 3)
04h	CPU	Arithmetic overflow trap
05h	CPU	Array bounds exception
06h	CPU	Invalid Opcode Trap
07h	CPU	Device Not Available Trap
08h	IRQ0	18.2 Hz Timer Tick
09h	IRQ1	Keyboard
0ah	IRQ2	Cascaded to PIC 2
0bh	IRQ3	COM2 Serial Port
0ch	IRQ4	COM1 Serial Port
0dh	IRQ5	LPT2 Parallel Port
0eh	IRQ6	Floppy Disk Controller
0fh	IRQ7	LPT1 Parallel Port
10h	Service	Video Services
11h	Service	Equipment List Service
12h	Service	Low Memory Size Service
13h	Service	Floppy/IDE/ROM/Remote Disk Services
14h	Service	Serial Port Services
15h	Service	General Services, Up-Calls
16h	Service	Keyboard Services
17h	Service	Parallel Port Services
18h	Up-Call	Boot Fault Up-Call
19h	Up-Call	Bootstrap Up-Call
1ah	Service	Date/Time Services
1bh	Up-Call	Control-Break Up-Call
1ch	Up-Call	18.2 Hz Application Timer Up-Call
1dh	Table	Pointer to Video Control Param Table
1eh	Table	Pointer to Diskette Parameter Table
1fh	Table	Pointer to Video Graphics Table
20h-3fh	DOS	-- reserved by DOS --
40h	Redirector	Floppy disk services redirected by IDE
41h	Table	Fixed Disk Parameter Table (Drive 80h)
42h	Extension	EGA Default Video Driver
43h	Extension	Video Graphics Characters
44h-45h	N/A	-- not used --
46h	Table	Fixed Disk Parameter Table (Drive 81h)
47h-49h	N/A	-- open --
4ah	Up-Call	User Alarm
4bh-6fh	N/A	-- open --
70h	IRQ8	Real-Time Clock Interrupt (1 Khz)
71h	IRQ9	-- open --
72h	IRQ10	-- open --
73h	IRQ11	-- open --
74h	IRQ12	PS/2 Mouse

75h	IRQ13	Math Coprocessor
76h	IRQ14	IDE Drive Controller
77h	IRQ15	APM Suspend Request
78h-ffh	N/A	-- open --

1.1.2.1 BIOS Service Interrupts

The BIOS receives requests to perform functions through software interrupts. Software interrupts, generated by the operating system or by a user application, are generated with INT *nn*h instructions, where *nn*h is a number that is assigned to a specific type of service, such as 16h for keyboard input, 10h for video output, or 13h for disk I/O.

In most cases, a BIOS service has multiple functions. For example, the disk BIOS service interrupt supports resetting the device, reading data from the media, writing data to the media, and checking the type of media inserted into the drive. For multifunction BIOS services, the requesting application places a function code in the AH CPU register, fills other registers as necessary with operands, and executes the appropriate software interrupt for the service. When the service completes, it returns to the caller to execute the instruction following the software interrupt.

Upon return, the BIOS services return status or other information in CPU registers, many times including the CPU flags register. For example, when an INT 13h disk read function is requested to read from a disk that has been removed from the drive itself, the disk BIOS returns with the carry flag set (CY) and a disk subsystem error code in the AH register. If the function were to complete successfully, then the carry flag would not be set (NC). Remember that not all BIOS services use the same return status conventions; therefore, you should consult the service reference in Chapter 22 for complete details.

1.1.2.1.1 INT 10h, Video Services

All video functions are provided through the INT 10h software interrupt mechanism. The caller provides a function code in the AH CPU register and specifies operands as appropriate for the given function in other CPU registers before issuing the INT 10h instruction.

EMBEDDED BIOS actually begins handling an INT 10h request in its CONIO module, which determines whether the video should be redirected over a serial link. This console redirection enables embedded systems that don't have a real MDA, CGA, EGA, or VGA video system to display their output via more inexpensive means. Console redirection may play a part in the final shipped embedded product, or it may simply be used during development and test in liu of an actual PC keyboard and screen.

If CONIO determines that the INT 10h service should not be redirected to a serial device, then it passes control to one of the modules that handle video controllers, such as module VIDEO, which manipulates the 6845 CRT controller registers directly to manage the display. Actual writing of data to the video screen and reading characters from the screen is accomplished by memory reads and writes to video regeneration memory, mapped into the memory address space at physical address 000b0000h for monochrome output, or 000b8000h for color output. Both monochrome and color adapters may be present in a system, in which case using the INT 10h set mode function can be used to switch between the displays.

If CONIO determines that INT 10h services should be redirected, then it calls the SERIAL module to perform the work of transmitting characters to the remote terminal equipment. In addition to writing characters to the display, the BIOS also supports the set cursor address function, and several other functions that manipulate the video display in some manner. These functions are translated to ANSI escape sequences that are transmitted to the remote terminal equipment just as other data characters via the SERIAL module's services through INT 14h.

The basic functions provided by the INT 10h BIOS are given below:

Function	Video Service
00h	Set Video Mode
01h	Set Cursor Type
02h	Set Cursor Position
03h	Return Cursor Position
04h	Return Light Pen Condition (not in core BIOS)
05h	Set Current Video Page
06h	Scroll Up Region
07h	Scroll Down Region
08h	Return Character and Attribute
09h	Write Character and Attribute
0ah	Write Character
0bh	Set Color Palette
0ch	Write Graphic Pixel (not in core BIOS)
0dh	Read Graphic Pixel (not in core BIOS)
0eh	Write Character Only
0fh	Return Video Display Mode

1.1.2.1.2 INT 11h, Equipment List Service

The BIOS provides a way for the application to determine what equipment is available through the INT 11h software interrupt mechanism. Unlike many of the other BIOS software interrupts, INT 11h does not require a function code or any operands. Instead, it returns a bit mask in its AX CPU register that can be inspected to determine what equipment is supported by BIOS services. For a complete description of this function, see Chapter 22.

The equipment list is stored in the BIOS Data Area (BDA) by POST during system initialization in a 16-bit field called **DevFlags**. ROM Extensions that extend BIOS services to support additional equipment must edit this field if the equipment is to be made available to the operating system or application.

1.1.2.1.3 INT 12h, Low Memory Size Service

The BIOS returns the amount of physical memory below the 1MB boundary (exclusive of the 1KB Extended BIOS Data Segment) in response to the INT 12h software interrupt. Like INT 11h (Equipment List), this software interrupt returns its information in the AX CPU register and does not accept function codes or operands. See Chapter 22 for full details.

The low memory size is stored in the BIOS Data Area (BDA) by POST during system initialization in a 16-bit field called **LowMemorySize**. ROM Extensions or other software that uses memory from the end of available low memory must reduce this field by the amount of memory reserved so that the operating system and applications will not overwrite the reserved memory. This technique is used by the BIOS itself during POST to establish the Extended BIOS Data Area (EBDA), a 1KB region located at the top of physical low memory.

1.1.2.1.4 INT 13h, Disk Services

All mass-storage devices, including floppy disk, hard disk, ROM disks, RAM disks, RFD disks, and OEM-defined disks, are accessed through the INT 13h software interrupt. As with the video services, INT 13h services accept a function code in the AH CPU register, with operands appropriate to a given function placed in the other CPU registers before executing the INT 13h instruction.

The following functions are supported by the FLOPPY disk driver (note that gaps in the function numbers indicate unassigned functions for floppy I/O):

Function	Floppy Disk Service
00h	Reset Floppy Controller
01h	Read Last Status
02h	Read Sectors
03h	Write Sectors
04h	Verify Sectors
05h	Format Track
08h	Read Drive Parameters
15h	Read Drive Type
16h	Determine Media Change
17h	Set Disk Type
18h	Set Media Type for Format

The following functions are supported by the IDE disk driver (note that gaps in the function numbers indicate unassigned functions for hard drive I/O):

Function	IDE Disk Service
00h	Reset IDE Controller
01h	Read Last Status
02h	Read Sectors
03h	Write Sectors
04h	Verify Sectors
05h	Format Track
08h	Read Drive Parameters
09h	Initialize Parameters
0ah	Read Long Sectors
0bh	Write Long Sectors
0ch	Seek to Cylinder
0dh	Alternate Reset
10h	Test Drive Ready
14h	Run Controller Diagnostic
15h	Read Disk Type

The following functions are supported by the ROM disk driver (note that write-oriented functions return a write-protected status for the ROM disk):

Function	ROM Disk Service
00h	Reset ROM Disk
01h	Read Last Status
02h	Read Sectors
04h	Verify Sectors
08h	Read Drive Parameters
15h	Read Drive Type
16h	Determine Media Change
18h	Set Media Type for Format

The following functions are supported by the RAM disk driver:

Function	RAM Disk Service
00h	Reset RAM Disk
01h	Read Last Status
02h	Read Sectors
03h	Write Sectors
04h	Verify Sectors
08h	Read Drive Parameters
15h	Read Drive Type
16h	Determine Media Change
18h	Set Media Type for Format

The following functions are supported by the Resident Flash Disk (RFD) driver:

Function	RFD Disk Service
00h	Reset Flash Disk
01h	Read Last Status
02h	Read Sectors
03h	Write Sectors
04h	Verify Sectors
08h	Read Drive Parameters
15h	Read Drive Type
16h	Determine Media Change
18h	Set Media Type for Format

Disk I/O is handled by different code modules in the BIOS, depending on whether a specific request is directed at a floppy device, an IDE hard drive, a ROM disk, a RAM disk, or a Resident Flash disk. During POST, the FLOPPY1/2/3, IDE1/2, ROMDISK, RAMDISK, and RFD1/2 modules are initialized if enabled through CMOS. POST maps these servers to specific drives when CMOS is scanned.

Disk I/O is logically divided into two types: floppy-compatible and hard drive-compatible. Traditionally, DOS requires floppy-compatible drives to have a FAT file system layout with a Partition Boot Record (PBR) in the first sector, two File Allocation Tables (FATs) following the PBR, and a root directory following the FATs. Hard drives are expected to be partitioned, and have a different logical layout. Starting with a Master Boot Record (MBR) in the first sector that contains a Partition Table, the remainder of the hard disk is divided into partitions that each have their own format logically similar to floppy disks. Each partition starts with a PBR, two FATs, and a root directory.

Because floppy disks and hard drives have different information organizations, the BIOS separates them into two sets of devices. Disks numbered 00h, 01h, 02h, and so on, are floppy drives, and DOS can expect floppy-style file systems on them. Disks numbered 80h, 81h, 82h, and so on, are hard drives, and DOS expects them to have an MBR, not a PBR, in the first sector.

The EMBEDDED BIOS ROM drive simulates one or more disks by treating the INT 13h read sectors function as simply a memory copy from OEM-specified areas of ROM to the application's data buffer. The memory image for each disk is created by the adaptation engineer using the DISKIMAG utility (provided with the EMBEDDED BIOS Adaptation Kit). ROM disks can be either soft (formatted like a floppy disk) or hard (formattted like a hard drive). The **FILE_SYSTEM** table entry in the project file that defines a ROM disk has a parameter that specifies whether the image is formatted as a floppy or a hard disk.

The EMBEDDED BIOS RAM drive is similar to the ROM drive, except that it supports both reading and writing. Build options specify the location of the RAM in the address space, and if automatic formatting is to be used by the BIOS during POST, in the event the RAM disk contents are not properly initialized.

The EMBEDDED BIOS Resident Flash Disk (RFD) operates only on Flash media, and can simulate both floppy disks and hard drives up to 32MB in size with a special wear-leveling algorithm that is built right into the core BIOS. Support for Flash media is provided through a Media Control Layer (MCL), which in turn calls Media Technology Drivers (MTDs) to perform the low-level I/O to the Flash. The **FILE_SYSTEM** table entry in the project file that defines a RFD has a parameter that specifies whether the image is formatted as a floppy or a hard disk.

The OEM can also implement special file system drivers and integrate them into the BIOS disk system without modifying the core BIOS source code. This is done by adding code to the board module, and then adding a special **FILE_SYSTEM** table entry in the project file that refers to the new file system's entrypoint.

1.1.2.1.5 INT 14h, Serial Port Services

All serial I/O functions are provided by the BIOS through the INT 14h software interrupt mechanism. As with disk drives, serial ports are numbered; the logical port numbers are 00h for COM1, 01h for COM2, 02h for COM3, and 03h for COM4.

The serial I/O service accepts a function code in the AH CPU register and operands in other registers. The logical port number is normally passed in the DX CPU register, so that the serial service can operate on a specific serial port. The following table shows a summary of the serial port services:

Function	Serial Port Service
00h	Initialize Serial Port
01h	Send Character
02h	Receive Character
03h	Read Port Status
04h	Extended Initialize
05h	Manipulate Modem Control Register

Upon return from the INT 14h instruction, status is returned in a complex way, with the AX CPU register containing both a Line Status Register and a Modem Status Register. Because this service exposes actual bit patterns used in the 8250, serial ports tied to incompatible UARTs (such as those on the 80C186-EC CPU) are supported by translating the status returned by such a UART into the the most-equivalent bitmask that would correspond to the 8250's status registers.

The BIOS handles INT 14h requests in the SERIAL module. This module translates logical port numbers into physical port numbers by indexing into the **ComPorts** array in the BIOS Data Area. Physical port numbers above 10h are assumed to be handled by the 8250 UART module, whereas port numbers 00h-10h are assumed to be handled by the CPU personality module. Thus, serial I/O requests are distributed on-the-fly to the appropriate hardware handler based on the configuration data in the BIOS Data Area.

The original IBM BIOS supported serial port data transfer rates through 9600 baud. The baud rate, as well as other communications parameters associated with a serial port, are configured using an INT 14h Initialize Serial Port function. EMBEDDED BIOS supports this standard as well as the Extended Initialize INT 14h function supported by modern desktop PC BIOS implementations, allowing higher baud rates through 115K baud.

1.1.2.1.6 INT 15h, General System Services

Often called a catch-all general service, the INT 15h software interrupt is actually used two ways: one where the application requests services, and another where the BIOS notifies the application that it is about to enter or leave a spin-loop in order to wait for a device to complete a task that will take some amount of real time. These "up-calls" or "call-outs" as they are sometimes called, interrupt the user application, which may choose to "hook" INT 15h to receive the notification, or not hook INT 15h, and therefore not be informed about the spin-loops. See the section on "BIOS Up-Calls" later in this chapter for further details.

The INT 15h services used by the application are implemented by the BIOS. These services are diverse; from returning the amount of available extended memory above 1MB, moving memory from one physical address to another, and switching into protected mode, to returning the address of the Extended BIOS Data Segment, returning the System Configuration Table (SCT) address, and manipulating the watchdog timer (see Chapter 22 for programming details). These services are all requested by placing a function code in the AH CPU register, setting other CPU registers to operand values, and executing an INT 15h instruction. Upon return, status is returned in several different ways; consult Chapter 21 for details. A summary of INT 15h services is shown in the following table.

Function	General Service
24h	Query A20 Port 92h Support
4fh	Scancode Translate Up-Call
53h	Advanced Power Management
85h	System Request Key Up-Call
86h	Wait Micro Interval
87h	Protected Mode Memory Block Move
88h	Return Extended Memory Size in KB
89h	Switch to Protected Mode
90h	Device Busy Up-Call
91h	Device Interrupt Up-Call

A0h	Read/Write CMOS Cell
A1h	Set Current I/O Redirection
A3h	Get Embedded BIOS Version
A4h	Query Embedded DOS-ROM file system
C0h	Return System Configuration
C1h	Return Extended BIOS Data Area
C2h	PS/2 Mouse
C3h	Enable/Disable Watchdog Timer
C4h	Checksum Memory Region
D0h	Breakpoint into BIOS Debugger
D8h	EISA Slot Configuration
E0h	Resident Flash Array Functions
FFh	Print Character for Embedded DOS-ROM Debug I/O

All INT 15h requests are dispatched by module MISC. Of course, if the operating system or application "hooks" IVT entry 15h so that it can receive up-calls, then it will also receive other function requests as well, and should pass them on to the BIOS in a "chained" approach.

Some functions are routed by MISC to the PROTMODE module, which handles steady-state protected mode processing in the BIOS. PROTMODE is complex because it must deal with several different mode switching techniques, and must also save the state of the CPU cache across mode switches. For details about what methods are available, consult Chapter 7.

1.1.2.1.7 INT 16h, Keyboard Services

All keyboard I/O functions are provided through the INT 16h software interrupt mechanism. Before executing an INT 16h instruction, the application places a function code in the AH CPU register, and other operands as appropriate in other CPU registers. Upon return from the INT 16h software interrupt, the status is returned in special ways, including through the Zero flag in the CPU. See Chapter 21 for programming details. A summary of INT 16h services is shown in the following table.

Function	Keyboard Service
00h	Read Character
01h	Return Keyboard Status
02h	Return Keyboard Flags
03h	Set Keyboard Typematic Rate
05h	Push Character/Scancode to Buffer
10h	Enhanced Read Character
11h	Enhanced Write Character
12h	Enhanced Return Keyboard Flags
f0h	Set CPU Speed
f1h	Get CPU Speed
f4h	Cache Control

As the table indicates, several keyboard functions in fact don't manipulate the keyboard. Instead, they manipulate other system components, such as the CPU's clocking and the system's cache. Because these features were commonly implemented in the 8042 keyboard controller of many desktop PC systems, their controlling BIOS functions were added to the INT 16h services. The CPU speed functions are routed to the HELPER module, and the cache control function is routed to the CACHE module.

As with video output through INT 10h, EMBEDDED BIOS is able to support a real PC or AT keyboard, or it can redirect INT 16h services over a serial port. Module CONIO receives the application INT 16h requests and determines how keyboard requests are to be serviced. If redirection to a serial port is enabled, then it calls the SERIAL module to read a character from a serial port or determine the serial port's status.

1.1.2.1.8 INT 17h, Parallel Port Services

All parallel port I/O services are provided through the INT 17h software interrupt mechanism. A function code is passed by the application in the AH CPU register, with additional operands as required in other CPU registers. In particular, the DX register is programmed with a logical printer port number, where 0=LPT1, 1=LPT2, and 2=LPT3. The following table shows the functions available in the INT 17h service family:

Function	Parallel Port Service
00h	Write Character
01h	Initialize Parallel Port
02h	Return Parallel Port Status

INT 17h requests are handled by the BIOS through module PARALLEL. This module translates the logical parallel port number into a physical port I/O address, and then manipulates that port directly to perform the function. Parallel port hardware is expected to be compatible with the IBM hardware.

1.1.2.1.9 INT 18h, Boot Fault Routine

After POST initializes the system, it calls INT 19h to boot the operating system from the appropriate device. If the INT 19h service fails to load the operating system, then the BIOS (or the operating system boot record) executes an INT 18h instruction, so that the ROM BIOS can regain control and perform an alternate function.

By default, EMBEDDED BIOS initializes the INT 18h function to a routine that prints "No boot device available.", and prompts to enter the debugger or SETUP system, or reboot the system.

At any point prior to the boot process, user-written code, such as code in ROM BIOS Extensions, can "hook" the INT 18h interrupt vector and gain control in this situation, thereby replacing the default handler in the BIOS. In the original PC, INT 18h jumped to a separate ROM that contained ROM BASIC. The embedded system developer might use this mechanism to execute application code from ROM in the event of a boot device failure.

1.1.2.1.10 INT 19h, Bootstrap Routine

After POST initializes the system, it calls module BOOTOS, which executes an INT 19h instruction to load the operating system or start the embedded application.

By default, EMBEDDED BIOS initializes the INT 19h function to a routine in module BOOTOS that cycles through six boot actions defined in CMOS cells. The six boot actions are attempted in order until one is successful. Keep in mind that, if ROM extensions such as DOS hook INT 19h so that they can get control when the system boots, then BOOTOS will not receive control to cycle through all of the boot actions, and the boot action sequence will be defeated.

Boot actions are Boot from any drive (A: through K:), Boot Windows CE in ROM, Boot Embedded DOS-ROM out of ROM, Enter Manufacturing Mode, Enter Debugger, and No Action.

The boot actions for drives A: through K: read one sector from the drive at sector 1, head 0, track 0 into physical memory location 00007C00h. If the read is successful and if the boot record contains the byte sequence 55h, aah, as the last two bytes in the 512-byte sector, then control is transferred to the boot record at 16:16 address 07C0:0000, and the BIOS plays no further role in the bootstrap process.

There are two ways to boot Windows CE with EMBEDDED BIOS. The first way is selected as a boot action "Boot Windows CE." The second way is by enabling a feature in SETUP that instructs EMBEDDED BIOS to attempt to find the Windows CE system file (NK.BIN) on disks that BOOTOS is told to boot from. If NK.BIN can be found during POST, it will be loaded into memory and booted. Otherwise, the boot record for that drive will be loaded and booted. This makes it possible to load Windows CE without loading DOS to run LOADCEPC. This feature of EMBEDDED BIOS is called "CE Ready."

As with the INT 18h interrupt vector, user-written code, such as code in a ROM BIOS Extension, can "hook" the INT 19h interrupt vector and gain control when it is time to load the operating system or start an embedded application. For systems with application code located and burned into ROM, the INT 19h vector can be hooked during POST and then be used as a way to receive control after the system has been initialized. This is how Embedded DOS-ROM receives control when it is configured to boot out of ROM.

1.1.2.1.11 INT 1ah, Time/Date Services

All time and date services are provided through the INT 1ah software interrupt mechanism. The application places a function code in the AH CPU register, and places any appropriate operands in other CPU registers, before executing an INT 1ah instruction. Upon return, INT 1ah services return their status in a complex way. The following table summarizes the available date/time services. Refer to Chapter 21 for complete programming details.

Function	Date/Time Service
00h	Return Ticks Since Midnight
01h	Set Ticks Since Midnight
02h	Return Time
03h	Set Time
04h	Return Date
05h	Set Date
b1h	PCI Services (architected by Intel)

The INT 1ah service manages the time and date as separate pieces of information, and in two ways. In systems with a PC-compatible Real Time Clock (RTC) component, the BIOS is capable of reading the contents of the RTC and updating it under program control. Both the date, and the time, can be stored in this device.

In systems that do not have a RTC component (and in those that do), the system time is maintained in a different way as a 32-bit number that represents "the number of ticks since midnight", in a location in the BIOS Data Area. It is common for DOS to detect whether the RTC services are available, and then use the ticks since midnight value as a system time when the real time clock is not present. When available, the RTC is normally the preferred method of obtaining the time, and is the only way of obtaining the date, since the RTC part is usually kept running with a battery when the system is turned off.

As can be seen from the table, INT 1ah is also the access point for PCI services, available on some targets. Consult Chapter 21 for details.

1.1.2.2 Table Pointers

Not all IVT entries point to a BIOS service routine. Several BIOS-managed interrupt vectors actually point to data structures maintained by the BIOS. These data structures are the Video Parameter Table (VPT), the Diskette Parameter Table (DPT), Video Graphics Character Table (VGCT) and the Fixed Disk Parameter Tables (VDPTs).

1.1.2.2.1 INT 1dh, Video Parameter Table (VPT)

The Video Parameter Table (VPT) is used by the VIDEO module to program the 6845 CRT controller's internal registers according to the specific mode requested by the application. The VPT is pointed to by IVT entry 1dh, and may be changed by software such as a VGA ROM Extension that supports additional modes.

The default VPT used by the integrated VIDEO module is shown below (this table is found in module BIOS in the source code):

```
;       The following table contains parameters (indexed by the user mode)
;       to load into the 6845's 16 operating registers.  Vector 1dh points
;       to this table, and the user software may replace it.

        PUBLIC  VideoTbl
VideoTbl label  byte
        db      38h, 40, 2dh, 10, 1fh, 6, 19h, 1ch, 2, 7, 6, 7, 0, 0, 0, 0
        db      71h, 80, 5ah, 10, 1fh, 6, 19h, 1ch, 2, 7, 6, 7, 0, 0, 0, 0
        db      38h, 40, 2dh, 10, 7fh, 6, 64h, 70h, 2, 1, 6, 7, 0, 0, 0, 0
        db      61h, 80, 52h, 15, 19h, 6, 19h, 19h, 2, 13, 11, 12, 0, 0, 0, 0
```

1.1.2.2.2 INT 1eh, Floppy Diskette Parameter Table (DPT)

IVT entry 1eh points to the current Diskette Parameter Table, or DPT, being used by the floppy disk BIOS. Because there are potentially several floppy drives in a system, the DPT defines the operational characteristics of the floppy currently being accessed.

The DPT pointer in the IVT is used by more than just the FLOPPY module. During the initialization of DOS, it copies the ROM-based DPT established by the BIOS into its own RAM buffer, and re-points the 1eh vector to the RAM location. This allows it to modify the default DPT before performing diskette operations so that they can be optimized.

Reestablishing the DPT in RAM serves another purpose as well. Softguard copy-protection relies on the fact that the DPT is copied into RAM by DOS, and edits the DPT pointed to by the 1eh vector to tell the BIOS that it will be reading 128-byte sectors during its check for a special, 128-byte sector on a certain track of a release diskette. When it is done calling INT 13h services to verify that this sector exists, then it restores the DPT to its original state.

Clearly, the DPT is not an architecturally sound way of providing more control over floppy disk services provided by the BIOS. Unfortunately, this architectural relic was firmly established with the first IBM PC BIOS and must be provided in other BIOS products.

The format of the DPT is shown below in an assembly language structure. This structure can be found in your INC directory in the STRUC.INC header file.

```
;       Diskette parameter table structure format.

DPT             struc
dpt_specify1    db      ?               ; specify command 1.
dpt_specify2    db      ?               ; specify command 2.
dpt_motoroff    db      ?               ; motor off time.
dpt_bps         db      ?               ; bytes per sector (coded, above).
dpt_spt         db      ?               ; sectors per track.
dpt_gap         db      ?               ; gap length between sectors.
dpt_dtl         db      ?               ; data length (always ffh).
dpt_gap3        db      ?               ; gap length for FORMAT.
dpt_fill        db      ?               ; fill byte for FORMAT.
dpt_headsettle  db      ?               ; head settle time.
dpt_motoron     db      ?               ; motor-on start time.
dpt_maxtrack    db      ?               ; max track number for this drive.
dpt_drr         db      ?               ; data transfer rate.
dpt_unused1     db      ?               ; unused byte.
dpt_unused2     db      ?               ; unused byte.
DPT             ends
```

The fields in the DPT are actually used as operand bytes when the FLOPPY1, FLOPPY2, and FLOPPY3 modules send commands to the Intel 82077A or 82078-compatible floppy disk controller. The specific values for each field are governed by the established standards for recording information on DOS-compatible floppy disks for the various drive types and media types. By manipulating these fields, the application program can cause the BIOS to read, write, format, and verify nonstandard media. For exact specifications on the values to be stored in the DPT, consult the Intel documentation on the 82077A or 82078 floppy disk controllers.

1.1.2.2.3 INT 1fh, Video Graphics Character Table (VGCT)

The Video Graphics Character Table (VGCT) is pointed to by IVT entry 1fh, and is used by VGA ROM BIOS Extensions to define the shape of the IBM-compatible character set when in graphics modes. When in character modes, the built-in VIDEO module in the BIOS does not use this entry. If you are internationalizing your adaptation of EMBEDDED BIOS to foreign character sets, this is the table to change the fonts for standard BIOS resolutions.

1.1.2.2.4 INT 41h/46h, Fixed Disk Paramter Tables (FDPTs)

IVT entries 41h and 46h are used in versions of the BIOS that support IDE drives, so that operating system software can determine the fixed disk drive types. Introduced by IBM with the IBM PC/AT Personal Computer, IVT entry 41h points to a data structure that describes the primary hard drive (drive 80h) and IVT entry 46h points to a data structure that describes the secondary hard drive (drive 81h). These structures should not be used by application software, and are rarely used by operating system software, since INT 13h function 08h can provide substantially the same information about both floppies and fixed disks in the system.

To maintain compatibility with the IBM PC/AT BIOS, EMBEDDED BIOS establishes these vectors to point to structures with the following format (see module IDE1 for how they are created):

```
FDPT         STRUC
fdpt_cyl     dw    ?            ; maximum number of cylinders.
fdpt_hd      db    ?            ; maximum number of heads.
             dw    ?            ; reserved, MBZ (not used, see PC/XT).
fdpt_wp      dw    ?            ; starting cyl for write precompensation.
             db    ?            ; reserved, MB. (max ECC data burst length).
             db    ?            ; DTE_CONTROL.
             db    ?            ; reserved, MBZ.
fdpt_cap     dw    ?            ; disk capacity in megabytes.
fdpt_lz      dw    ?            ; landing zone cylinder.
fdpt_spt     db    ?            ; sectors per track.
             db    ?            ; reserved.
FDPT         ENDS
```

1.1.2.3 BIOS Upcalls

While nearly all of the software interrupts associated with the BIOS are invoked by the operating system or application and serviced by the BIOS itself, there are a few software interrupts that are actually generated by the BIOS, and may be "hooked" by the operating system or the application. These software interrupts, called "up-calls" or "call-outs", are used to notify application software that events in the BIOS have occurred.

1.1.2.3.1 INT 15h Device Management

The INT 15h software interrupt is a two-way interrupt service. Functions such as 87h, 88h, and 89h are made by the application and serviced by the BIOS to provide protected mode support. Other functions, such as 90h and 91h, are invoked by the BIOS, and "hooked" by DOS or by application software to be notified when events inside the BIOS occur. These invocations of INT 15h functions by the BIOS are called "up-calls", or simply, "call-outs".

INT 15h up-calls are generated by various modules within the BIOS. The floppy and hard disk modules are the most important ones, as they involve comparatively large intervals of time during head seeks and waiting for rotational latency of the media. During seeks and disk rotations, an operating system can use INT 15h to gain control and perform other tasks until notified that the operation has completed. Just as with the other INT 15h services, the BIOS places a function code in the AH CPU register, with a device code in other registers, before executing its INT 15h instruction.

1.1.2.3.1.1 INT 15h Function 4fh

Function code 4fh is used to indicate that the keyboard has received a keypress or key release interrupt, with a scancode in the AL CPU register from the keyboard controller. The KEYBOARD module issues the INT 15h function to give the application a chance to interpret the scancode and modify it if required.

Upon return from the INT 15h function 4fh call, the keyboard BIOS checks the state of the carry flag. If the carry flag is cleared by the application, then the BIOS performs no more processing on the scan code and assumes that the application handled it. If the carry flag was set by the application, then the BIOS handles the scan code as returned by the application in the AL CPU register. The latter case allows the application to modify the scan code in the AL register without handling it directly. Because the BIOS sets the carry flag before issuing the INT 15h instruction, the BIOS will handle the scan code by default if the application does not modify the carry flag.

1.1.2.3.1.2 INT 15h Function 90h

Function code 90h is used to indicate that a spin-loop is about to be executed by a BIOS component. When the BIOS invokes INT 15h function 90h, it passes a device code in the AL CPU register that indicates what device is causing the wait. The following device codes are architected by IBM:

Code	Device Name
00h	IDE Hard Drive
01h	Floppy Disk Drive
02h	Keyboard
03h	PS/2 Mouse
80h	Network
FCh	Hard Disk Reset Operation
FDh	Floppy Disk Drive Motor Control Operation
FEh	Printer

Upon return, the application software that hooks the INT 15h function 90h service should set the CY flag if it did not wait for the device to complete its operation, or clear the CY flag if a wait or timeout occurred in the application code. The state of the CY flag is tested by the BIOS when the INT 15h function 90h routine returns to determine whether to actually perform or skip the spin-loop.

1.1.2.3.1.3 INT 15h Function 91h

Function code 91h is used to indicate that a device interrupt has just been received that would complete the spin-loop. As with function 90h, a device code is passed in the AL CPU register to indicate which device has just completed an operation. The device codes for functions 90h and 91h are identical.

Upon return, the application software that hooks the INT 15h function 91h service should set the AH CPU register to 00h and clear the CY flag.

1.1.2.3.1.4 INT 15h Function 85h

Another INT 15h up-call provides notification that the user has pressed or released the **SysReq** key on an AT-class (101-key) keyboard. When the KEYBOARD.ASM module detects that this key is pressed, it issues an INT 15h with AH=85h, and sets the AL CPU register to 00h, indicating that they key was depressed. When the key is released, it issues an INT 15h with AH=85h, and sets the AL CPU register to 01h.

1.1.2.3.2 INT 1bh Control-Break Signal

The KEYBOARD.ASM module executes an INT 1bh software interrupt if it detects that the user pressed the Control and Break keys simultaneously. This allows the application to gain control when this happens. Upon return from the INT 1bh instruction, the BIOS stores a 00h scan code and 00h character code in the keyboard's typeahead buffer.

DOS normally hooks the INT 1bh Interrupt Vector Table entry so that it can terminate a program prematurely. The mechanisms used by DOS to make this happen are proprietary to the specific version of DOS and are beyond the scope of this manual.

1.1.2.3.3 INT 1ch User Timer Interrupt

The BIOS provides a regular 18.2Hz heartbeat for operating systems and applications by executing an INT 1ch instruction every 55 milliseconds. By default, the BIOS has its own INT 1ch handler that does nothing, so that the application software is not required to provide a handler unless one is needed.

In strictly ISA systems, INT 1ch is executed inside the IRQ0 Interrupt Service Routine of the BIOS after the 8254 Programmable Interval Timer's T0 timer expires, and after the End-Of-Interrupt (EOI) has been issued to the primary Programmable Interrupt Controller (PIC). Thus, suspension inside the INT 1ch handler by the application does not degrade system performance the way it would be if the application suspended operations inside the INT 08h handler.

In non-ISA systems, such as those designed around the NEC V-Series (i.e., V25) processors, EMBEDDED BIOS cannot program the on-board timers to generate an interrupt on INT 08h. Instead, the timer is actually hard-wired to interrupt vector 1ch. The BIOS accounts for this and calls INT 08h inside the INT 1ch handler. Application software should be aware of this possibility and not block inside the INT 1ch handler. Instead, they should chain the INT 1ch handler, call the lower layer first, and then perform any work as required. This technique gives the BIOS a chance to issue an EOI before any application code runs.

1.1.2.3.4 INT 4ah Real Time Software Interrupt

Just as the ISA IRQ0 hardware timer interrupt routed to INT 08h causes the INT 1ch user timer software interrupt to be generated every 55ms, ISA IRQ8 is tied to a 1Khz timer routed to INT 70h, which in turn causes an INT 4ah instruction to be generated every 1ms (at a 1Khz rate).

Commonly called the real-time clock interrupt, INT 4ah can be "hooked" by real-time kernels to gain control for rescheduling purposes on ISA platforms. Warning: Nonstandard platforms may not provide this support, as it is provided by the Dallas Real-Time Clock (RTC) chip in PC/AT-compatible targets.

To enable the 1Khz INT 4ah interrupt heartbeat, the operating system or application must manipulate the CMOS RTC registers. The BIOS automatically routes the hardware interrupt (IRQ8) to interrupt vector 70h. The BIOS-supplied ISR for INT 70h then calls INT 4ah after issuing an EOI to both Programmable Interrupt Controllers (PICs).

1.1.2.4 CPU Traps/Faults

Intel 8086-family processors and their architectural equivalents all provide a way for the operating system or application program to gain control when an instruction cannot be executed for some reason. When the CPU encounters a problem with executing an instruction, it generates an exception.

When EMBEDDED BIOS is built with the option to enable the integrated BIOS debugger, the BIOS routes all the CPU-generated exceptions to the debugger itself, so that the adaptation engineer can determine why the exception occurred and then debug the problem. Without the debugger enabled, the operating system or application program is responsible for catching exceptions and handling them in an appropriate manner.

There are two types of exceptions; namely, traps and faults. Traps are generated when something happens that makes it impossible for the instruction to be restarted. When an invalid instruction is detected, for example, an "Invalid Instruction Trap" occurs.

Faults are different from traps in that a fault handler can perform some sort of work that would potentially allow the problem instruction to be able to re-execute correctly. A good example of such a fault is the "Page Fault" mechanism commonly used in virtual memory management systems in protected-mode operating systems. Because an instruction may execute from a page that is not present in memory, or perhaps because its operands in memory are located in pages of memory that are not present, the page fault mechanism gives the operating system control so that the necessary pages of virtual memory can be mapped to real physical memory. Once the mapping is completed, the fault routine returns to the interrupted context, and the instruction proceeds as though the fault never happened.

In Intel CPUs, the following exceptions can be generated by the CPU. Note that some are marked as traps, and some are faults. Also note that the interrupt vector numbers assigned to the exceptions conflict with the BIOS service interrupt numbers. This is not a misprint; it is an historical part of the BIOS architecture first defined by IBM.

Vector	Processors	Exception Type
00h	8086	Divide Error Trap
01h	8086	Instruction Trace Trap
02h	8086	NMI Interrupt (Trap)
03h	8086	Breakpoint Trap
04h	8086	Arithmetic Overflow Trap (INTO Instructions)
05h	80286	Array Bounds Trap
06h	8086	Invalid Opcode Trap
07h	8086	Device Not Available Trap
08h	80286	Double Fault
09h	N/A	-- Reserved for Future Use --
0ah	80286	Invalid Task State Segment Fault
0bh	80286	Segment Not Present Fault
0ch	80286	Stack Exception Fault

0dh	80286	General Protection Fault
0eh	80386	Page Fault
0fh	N/A	-- Reserved for Future Use --
10h	80386	Floating Point Fault
11h	80486	Alignment Fault
12h	80486	Machine Check Fault

1.1.2.5 Hardware Interrupts

EMBEDDED BIOS is configurable to support a wide variety of processors that provide at least the functionality of the Intel 8088 CPU. Processors in the Intel 8086 family include the 80286, the 80386, the i486, Pentium, and Pentium-Pro CPUs, and these CPUs are generally deployed in ISA, PCI, or local bus-type system architectures.

Intel's 80C186-EA/EB/EC family of processors provide a superset of the instruction set, but in addition have on-board peripherals that are not like those in an ISA-class machine. Instead, the on-board timers, serial ports, and so on, are internally wired to different interrupt request lines, which in turn translate to different interrupt vectors that must be serviced by the BIOS.

Similarly, the NEC V-Series processors execute supersets of the Intel 8086 CPU's instruction set; however, their on-board peripherals are also proprietary and are internally-wired to different interrupt request lines. These different IRQs are necessarily routed through different interrupt vectors.

ISA-class systems all have a similar interrupt model for hardware interrupts. The ISA interrupt assignments are as follows:

IRQ	Vector	Device
IRQ0	08h	8254 Programmable Interval Timer
IRQ1	09h	Keyboard Controller
IRQ2	0ah	Cascade Interrupt to PIC2
IRQ3	0bh	COM2 Serial Port (8250)
IRQ4	0ch	COM1 Serial Port (8250)
IRQ5	0dh	LPT2 Parallel Port
IRQ6	0eh	Floppy Disk Controller
IRQ7	0fh	LPT1 Parallel Port
IRQ8	70h	Real-Time Clock Interrupt (1Khz)
IRQ9	71h	-- open --
IRQ10	72h	-- open --
IRQ11	73h	-- open --
IRQ12	74h	PS/2 Mouse
IRQ13	75h	Math Coprocessor
IRQ14	76h	IDE Drive Controller
IRQ15	77h	-- open* --

* Caution: Some BIOS implementations may use INT 77h as a software suspend request in their Power Management module. Operating systems or applications execute the INT 77h, which is received by the BIOS as a request to power-down the system.

1.10.3 System Configuration Table

BIOS service INT 15h function C0h returns a pointer to a data structure called the System Configuration Table (SCT), an area inspected by DOS and applications to determine which features are supported by the underlying BIOS. The SCT is defined in module BIOS and may be modified by the adaptation engineer.

The contents of the standard SCT are given below:

```
        PUBLIC  SCT
SCT:
        dw      SCT_End - SCT - 2
        db      BIOS_HDWR               ; hardware ID byte.
        db      BIOS_MAJOR_VERSION
        db      BIOS_MINOR_VERSION

;       The next byte contains bitflags as follows, indicating what
;       features the BIOS supports.
;
;       bit 7 - BIOS using DMA ch3
;       bit 6 - cascaded IRQ2
;       bit 5 - real-time clock present
;       bit 4 - int 1Ah is keyboard scan

SCTFLAGS =      00000000B               ; start out with nothing.

        IF      OPTION_SUPPORT_8259_2
SCTFLAGS =      SCTFLAGS OR 01000000B   ; bit 6 - cascaded IRQ2.
        ENDIF   ; (OPTION_SUPPORT_8259_2

        IF      OPTION_SUPPORT_CMOS
SCTFLAGS =      SCTFLAGS OR 00100000B   ; bit 5 - real-time clock present.
        ENDIF   ; (OPTION_SUPPORT_8259_2)

SCTFLAGS =      SCTFLAGS OR 00010000B   ; bit 4 - INT 1Ah is keyboard scan.

        db      SCTFLAGS                ; define flag byte with above bitflags.

        db      4 dup(0)                ; reserved
SCT_End EQU     $
```

1.11 Console I/O Redirection

Both INT 10h (video) and INT 16h (keyboard) services may be redirected by EMBEDDED BIOS to any serial port, so that the application, Setup screen, and integrated BIOS debugger can all communicate over an RS-232 link to a remote host running a terminal program. These three classes of I/O can be redirected independently, to any valid serial port in the system supported by the `SERIAL.ASM` module.

1.11.1 Video (INT 10h) Redirection

Redirection of INT 10h (video) requests happens in CONIO by looking at the **CurrIo** field in the Extended BIOS Data Area (EBDA). If this field is set to **IO_CONSOLE** (0), then video is routed to the VIDEO module, which programs the 6845 CRT controller. If this field is set to **IO_COM1** (1), **IO_COM2** (2), **IO_COM3** (3), or **IO_COM4** (4), then the I/O is redirected to the specified port. **IO_NONE** is a value the system uses to disable INT 10h output altogether.

Other fields in the EBDA take part in redirection. Consider the fact that application INT 10h services are separated from debugger INT 10h services and Setup screen INT 10h services. This is handled by the Setup and Debugger modules by setting the **CurrIo** field to the values in **SetupIo** or **DebugIo**, respectively, before those modules do any output. Then, when the modules are finished with their processing, they restore the **CurrIo** field to its former value. The save areas for this restoration are **SetupIox** and **DebugIox**, respectively. An INT 15h function is available for handling these details; it is callable from the application program, or from the Board, CPU, or Chipset Personality Modules if a stack is available.

1.11.2 Keyboard (INT 16h) Redirection

Redirection of INT 16h requests to the `SERIAL.ASM` module happens in module `CONIO.ASM` by looking at the **CurrIo** field in the Extended BIOS Data Area (EBDA). If this field is set to **IO_CONSOLE** (0), then keyboard requests are passed to module `KEYBOARD.ASM`, which programs the 8042 keyboard controller on an AT, or the 8255 peripheral interface on a PC/XT compatible machine. If this field is set to **IO_COM1** (1), **IO_COM2** (2), **IO_COM3** (3), or **IO_COM4** (4), then the I/O is redirected to the specified port. **IO_NONE** is a value the system uses to disable INT 16h output altogether.

Just as with INT 10h services, other fields in the EBDA take part in input redirection. Because application INT 16h services are separated from debugger INT 16h services and Setup screen INT 16h services, the Setup and Debugger modules set the **CurrIo** field to the values in **SetupIo** or **DebugIo**, respectively, before those modules request any input. Then, when the modules are finished with their processing, they restore the **CurrIo** field to its former value. The save areas for this restoration are **SetupIox** and **DebugIox**, respectively.

Module `SERIAL.ASM` doesn't actually do the I/O directly for the I/O redirection. Instead, it translates the logical serial port number associated with the console redirection into a physical port number, and then calls the 8250 driver module, or the CPU personality module, depending on where the physical UART is located.

1.12 Integrated BIOS Debugger

When bringing-up new hardware, it is essential to have a debugging tool that can disassemble code, display and alter the contents of memory, write to I/O ports, breakpoint code, and test the operation of the A20 line and CMOS storage. These functions are all features of the integrated BIOS debugger that is provided with EMBEDDED BIOS.

By enabling the **OPTION_SUPPORT_DEBUGGER** configuration option in `CONFIG.INC`, the debugger code will be automatically assembled into the BIOS. Then, when the system boots, the debugger can be started in several ways.

First, on machines with a PC or PC/AT-compatible keyboard, the debugger can be entered through a special key chord. Just depress both the left ALT key and the left SHIFT key to break into the debugger.

Second, the debugger hooks the CPU exception vectors in case a divide by zero occurs, an invalid opcode is executed, or an INT 3 instruction is executed, for example. By placing an INT 3 in the POST mainline code (or anywhere else in the BIOS source code) after INT 10h and INT 16h services are available, the debugger will automatically be invoked. To resume, type the 'G' command to "GO", or continue on with the rest of initialization.

Third, the debugger can be entered from the Setup main menu, if the debugger Setup screen option is enabled. This allows an end-user to access the integrated BIOS debugger from within the full-screen menuing system.

Fourth, the debugger is a selectable boot action, allowing it to gain control if any of the other bootable drives are not available or are not formatted. This is controlled via the Basic SETUP screen.

Finally, the debugger can be entered if no operating system can be loaded. The system displays a message that indicates that a boot device cannot be found, and then prompts the user to press the ESC key to enter the debugger.

The debugger can be used over a serial port, in the event that the target system has no keyboard or monitor, or if those devices are being used by the application. For example, if a graphics application has drawn on the screen, the integrated BIOS debugger's output would disrupt the video display if it were not redirected. Redirection of debugger output is controlled via the **OPTION_CONIO_DEBUG** configuration option in the project file.

Use of the integrated BIOS debugger is outside the scope of this section; consult Chapter 9 for complete details.

1.18 Protected Mode Support

On 80386 and above processors, EMBEDDED BIOS uses the protected mode of the CPU to access extended memory. To configure EMBEDDED BIOS to support protected mode, the **OPTION_SUPPORT_PROTMODE** option must be enabled. Support for the 80286 CPU has been discontinued due to the CPU's architectural limitations and end-of-life.

When configured, protected mode is used during POST and during steady-state operation of the system. During POST, the BIOS determines the amount of extended memory available by performing memory tests in protected mode. During steady-state, the operating system or application program can request that the BIOS switch to protected mode with INT 15h function 89h, and move memory while in protected mode with INT 15h function 87h.

Protected mode support is complicated by the various ways in which the processor can be instructed to resume execution in real mode after a protected mode operation. In 80286-based systems, there was no "switch to real mode" CPU instruction. Therefore, these systems had an outboard hardware solution that involves any of several components: the 8042 keyboard controller, I/O port 92h, or the chipset. These methods are supported by EMBEDDED BIOS in the event that systems continue to use them even though they are not 80286-based.

On 80386 and above systems, the CPU contains a "`MOV CR0, EAX`" instruction that allows the BIOS to return to real mode without the use of external hardware. The adaptation engineer can select that this option be used when it is known that the target will be using an 80386 or better CPU.

It remains common for the outboard hardware reset techniques to be used in designs that use 80386 or better CPUs, simply because these features have been added to chipsets for full compatibility with the IBM PC/AT standard machine. Thus, the adaptation engineer should review the methods by which the target hardware can be switched back into real mode, and select the one with the lowest overhead.

Chapter 2

THE INTEGRATED BIOS DEBUGGER

This chapter describes the operation of the *integrated BIOS debugger*. The purpose of the debugger is to provide BIOS-level debugging facilities to the BIOS customization and hardware development team and to the operating system and application engineers involved in bringing up the operating system or application software on the target. It is not intended for use as the only debugging tool for application programmers; it is mainly used for ensuring that the application software or source-level debugger is loaded properly in the system, and that all the BIOS services are available to the higher layer software.

2.1 How to Use the Debugger

The debugger can be activated in four ways.

1. On a PC-compatible platform, the BIOS debugger can be invoked through the console by pressing the keyboard's CTRL and LEFT-SHIFT keys simultaneously. Breaking into the debugger in this way suspends the execution stream in the system until it is resumed with the "G" (go) command. As part of the standard configuration options, the OEM can configure the debugger to communicate through a serial port rather than the console, if desired.

2. The debugger can also receive programmed control from an "**INT 3**" instruction in the DOS code, application code, or BIOS code. This can be useful in debugging EMBEDDED BIOS adaptations running on new hardware that aren't yet booting the operating system. It can also be used to check-out new hardware by manipulating I/O ports with debugger commands.

3. From the SETUP main menu, the **ENTER SYSTEM BIOS DEBUGGER** selection will enter the debugger. After use, typing the "G" (go) command will return to the SETUP screens.

4. As a boot action, as a last-ditch effort if the operating system cannot be booted from the appropriate drives or out of ROM, and the Manufacturing Mode link cannot be established.

To enable the debugger in your EMBEDDED BIOS adaptation, set **OPTION_SUPPORT_DEBUGGER** to 1. The debugger contains quite a few commands and also contains a disassembler, which includes a full opcode table (see related options of the form,

OPTION_DEBUG_*xxx*.) This can take up quite a bit of space, and it may require that you increase the size of the BIOS itself by increasing **OPTION_BIOS_KBSIZE** to 64.

To enable access to the debugger through the SETUP screen system, enable **OPTION_SETUP_DEBUGGER**.

As with the SETUP system, the debugger can be configured to redirect its input and output over an OEM-defined serial port. To redirect debugger I/O over an RS232 serial line, enable **OPTION_SUPPORT_CON_REDIRECTOR**, and set **CONFIG_CON_REDIR_DEBUG** to the serial port number (1=COM1, 2=COM2, etc.) to use for remote debugging.

2.2 Debugger Command Syntax

Nearly all debugger commands are specified as a single abbreviated word such as "BIOSDATA", "REBOOT", or "G", followed possibly by expressions separated by tabs, spaces, or commas. Depending on the command's function, the address operand may default to the "next approriate address" or it may be required in the event that there is no "next appropriate address". Command names are case-insensitive, as are the names of registers in operands.

2.2.1 Operand Types

Commands all take different operand types, depending on their function. For example, the command that outputs a 32-bit double word to a 32-bit I/O port requires a 32-bit datum, whereas the command that dumps memory uses an address of the memory area to dump.

The debugger accepts 8-bit, 16-bit, and 32-bit operands, as needed for a given command. In addition, real-mode (segment:offset) addresses, and 32-bit physical addresses, are often given. The Flash programming commands require a 32-bit address formed by an xxxx:yyyy syntax that looks like it should be a real-mode address, but is in fact a special way to enter a 32-bit physical Flash media address in two components, separated by a semicolon.

The 8-bit, 16-bit, and 32-bit operands are built from expressions.

Real-mode addresses (often called 16:16 addresses) are composed of two 16-bit expressions separated by a colon, where the 16-bit expression on the left hand side of the colon represents the segment, and the 16-bit expression on the right hand side represents the offset.

Physical addresses are indicated by a percent sign (%) followed by a 32-bit expression.

2.2.2 Expressions

The basic components of expressions are simple values, such as hexadecimal constants, or register names. For example, the constants 0, 1234, 17, DEADBEEF (an interesting-looking 32-bit hexadecimal number), and register names AX, BX, CX, DX, SI, DI, SP, BP, FL, CS, DS, ES, SS, FS, and GS are all simple values. The 8-bit register names are not valid simple values. The 32-bit register names EAX, EBX, ECX, EDX, ESI, EDI, ESP, and EBP, are also valid when the **CONFIG_CPU_TYPE** parameter has been set to **CPU_386** or above. Whenever an expression is called for, these values can be used alone.

In addition, parentheses may be used (**without any intermixed white space**) to specify a simple expression consisting of an operation to be performed on two values. For example, it is possible to take the sum, difference, logical AND, or logical OR of two values, or shift one value by the number of bits specified by the second value. Finally, anywhere a value may be specified, a simple expression may be specified.

This recursive definition leads to the following examples of 16-bit expressions that might be used in debugger operands:

`1234`	(constant value)
`BX`	(contents of BX general register)
`(BX+1234)`	(contents of BX plus 1234h)
`(AX-2345)`	(contents of AX minus 2345h)
`(CX&55AA)`	(contents of CX ANDed with 55aah)
`(BP|2)`	(contents of BP ORed with 0002h)
`(FL>1)`	(contents of FL shifted right one bit)
`(ES<AX)`	(contents of ES shifted left AX bits)

Here are some more complex 16-bit expressions:

`(BX+(SI-23))`	(add BX to the difference of SI and 23h)
`((AX&7FFF)|(BX&8000))`	(bottom 15 bits of AX ORed with top bit of BX)

Here is a more formal definition, using a modified BNF, of expression syntax:

```
<hex value>         ::=   <hex digit> | <hex digit> <hex value>

<reg16>             ::=   AX | BX | CX | DX | SI | DI | SP |
                          BP |  DS | ES | CS | SS | FS | GS | FL

<reg32>             ::=   EAX | EBX | ECX | EDX |
                          ESI | EDI | ESP | EBP | EFL

<register>          ::=   <reg32> | <reg16>

<value>             ::=   <hex value> | <register>

<operator>          ::=   '+' | '-' | '&' | '|' | '>' | '<'

<expr>              ::=   <value> | ( <expr> <operator> <expr> )
```

2.2.3 Addresses

Some debugger commands, such as U[nassemble] and D[ump bytes], allow an address to be specified. When this is the case, the address can be a real-mode address or a physical address.

Real-mode addresses consist of two 16-bit expressions separated by a colon without intervening whitespace. For example, F000:1234 specifies offset 1234h with respect to real mode segment F000h. Register names, and expressions involving constants and expressions, are supported. For example, F000:(BX+52) specifies an offset calculated from the contents of the BX CPU register summed with the hexadecimal constant, 52h.

Physical addresses are specified by a 32-bit expression prefixed by a percent sign (%). The following are examples of 32-bit physical addresses:

`%00800000`	(address of first byte at 8MB boundary)
`%00100000`	(address of first byte of extended memory)
`%00000000`	(address of first byte of low memory)
`%FFFFFFFF`	(address of last byte in Pentium-class machine)

Flash media commands use a special form of addressing to indicate 32-bit physical addresses. This form consists of two 16-bit expressions separated by a colon, as with real-mode addresses. However, the two 16-bit expressions do not correspond to a segment:offset pair. Instead, the first 16-bit expression becomes the high 16 bits of the 32-bit address, and the second 16-bit expression becomes the low 16 bits of the 32-bit address. The following are examples of Flash addresses:

`0000:0000`	(address of first byte of low memory)
`0010:0000`	(address of first byte of extended memory)
`0080:0000`	(address of first byte at 8MB boundary)
`FFFF:FFFF`	(address of last byte in Pentium-class machine)

2.3 Command Reference

This section describes the individual debugger commands.

2.3.1 ? Command

The "?" command evaluates its operand as an expression and prints the resulting value.

Command Syntax:

`?` *Expression*

Parameters:

Expression - A required expression as specified earlier in this chapter.

Sample Output Display:

```
1234
```

2.3.2 + Command

The "+" command advances the instruction pointer (IP) by one byte. This command is useful when skipping over instructions.

Command Syntax:

```
+
```

Parameters:

none.

Sample Output Display:

none.

2.3.3 - Command

The "-" command backs up the instruction pointer (IP) by one byte. This command is useful when an instruction should be reexecuted.

Command Syntax:

```
-
```

Parameters:

none.

Sample Output Display:

none.

2.3.4 BC Command

The BC command allows the developer to clear an execution breakpoint.

Command Syntax:

`BC` *BreakpointNumber*

Parameters:

BreakpointNumber - A required expression parameter that specifies the number of the breakpoint to be cleared. The number of a breakpoint can be displayed with the BL command, and is displayed after the system processes a BP command.

Sample Output Display:

```
Breakpoint #0 cleared.
```

2.3.5 BIOSDATA Command

The BIOSDATA command allows the developer to inspect the major low-memory fields in the system at segment 40H.

Command Syntax:

```
BIOSDATA
```

Parameters:

none.

Sample Output Display:

```
Different for different BIOS adaptations
```

2.3.6 BL Command

The BL command allows the developer to list the breakpoints that are currently active. If a breakpoint has a command string associated with it, the command string is displayed. Those breakpoints with no command string have no command string display.

Command Syntax:

```
BL
```

Parameters:

none.

Sample Output Display:

```
#0 - 0500:1d49      “U CS:IP;G”
#1 - 12d9:ef7c      “CSW 32 1A;R AX 1234;T”
#2 - 12e9:459a
```

2.3.7 BP Command

The BP command allows the developer to set an execution breakpoint at a specified address. Multiple breakpoints may be set at any given time. Please note that breakpoints work by storing an INT 3 instruction at the specified location; this is impossible in read-only memory. Breakpoints may only be set in RAM.

Command Syntax:

`BP` *Address* [*“CommandString”*]

Parameters:

Address - A required parameter that specifies the 16:16 real-mode address of a breakpoint to be set.

CommandString - An optional parameter, enclosed in double quotes, that specifies a sequence of commands separated by semicolons to be executed when the breakpoint occurs. If this parameter is not specified, then the standard breakpoint command is the R (register dump) command.

Sample Output Display:

```
Breakpoint #0 saved.
```

2.3.8 CIS Command

The CIS command allows the developer to display a portion of the memory address space with the formatting of a Card Information Structure as found in the configuration space of PCMCIA cards.

This command is intended for use in debugging embedded applications that have a dedicated PCMCIA card that must be configured for use in the target without card or socket services.

Command Syntax:

`CIS` *Address*

Parameters:

Address - A required parameter that specifies the 16:16 real-mode address of memory space to be formatted as a CIS.

Sample Output Display:

Dependent on PCMCIA Card Type.

2.3.9 CONSOLE Command

The CONSOLE command allows the developer to redirect the debugger's input and output to another device. Available devices are:

CON - the system keyboard and video display monitor
COM1 - the first communications port at 3f8h
COM2 - the second communications port at 2f8h

Command Syntax:

`CONSOLE` *Device*

Parameters:

Device - A required parameter that specifies the new console to redirect debugger output to, and to redirect debugger input from.

Sample Output Display:

none.

2.3.10 CSR Command

The CSR ("chip set read") command allows the developer to display the value held in a chipset register. If no chipset is configured for the BIOS adaptation, then this command cannot function properly.

Note that some chipset registers are write-only, and some chipsets (or their equivalents on high-integration CPUs such as the SC300) may have registers that read-out different values than the values written to them (bits flip, and some may stay high or low.)

This command is very useful in conjunction with CSW to test chipset configuration values before building a new BIOS with best-guess values.

Command Syntax:

`CSR` *RegisterIndex*

Parameters:

RegisterIndex - A required expression that specifies the index of the register in the chipset to be read.

Sample Output Display:

```
1234h
```

2.3.11 CSW Command

The CSW ("chip set write") command allows the developer to set a chipset register to a specific value. If no chipset is configured for the BIOS adaptation, then this command cannot function properly.

Note that some chipset registers are write-only, and some chipsets (or their equivalents on high-integration CPUs such as the SC300) may have registers that read-out different values than the values written to them (bits flip, and some may stay high or low.)

This command is very useful in conjunction with CSR to test chipset configuration values before building a new BIOS with best-guess values.

Command Syntax:

`CSW` *RegisterIndex RegisterValue*

Parameters:

RegisterIndex - A required expression that specifies the index of the register in the chipset to be read.

RegisterValue - A required expression that specifies the value to be stored in the chipset register. Note that some chipsets use 8-bit values, and others use 16-bit values. See your chipset's programming documentation for details.

Sample Output Display:

None.

2.3.12 D Command

The D command allows the developer to display memory at the specified address, or at the address immediately following the last byte displayed with the last D, DB, DW, or DD command.

Command Syntax:

`D` *Address*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode or 0:32 physical address of memory to be displayed in the default format. If not specified, then the address is assumed to be the address immediately following the last byte displayed by the last D, DB, DW, or DD command.

Sample Output Display:

```
0040:0000 f8 03 00 00 00 00 00 00:bc 03 78 03 00 00 00 00  o.......L.x.....
0040:0010 7d 82 00 80 02 00 00 00:00 00 2c 00 2c 00 13 1f  }e.C......,.,...
0040:0020 13 1f 3f 35 0d 1c 03 2e:64 20 0d 1c 6f 18 75 16  ..>5....d...o.u.
0040:0030 74 14 0d 1c 62 30 6c 26:0d 1c 3f 35 0d 1c 01 00  t...b0l&..?5....
0040:0040 24 00 04 00 00 00 01 06:02 07 50 00 00 40 00 00  #.........P..@..
0040:0050 0b 18 00 00 00 00 00 00:00 00 00 00 00 00 00 00  <...............
0040:0060 07 00 00 b4 03 29 30 03:00 00 c8 00 b1 93 01 00  .....)0......o..
0040:0070 00 00 00 00 00 01 81 00:14 14 14 14 01 01 01 01  ......u.........
```

2.3.13 DA20 Command

The DA20 command allows the developer disable the A20 gate using the method configured in the BIOS adaptation.

Command Syntax:

```
DA20
```

Parameters:

none.

Sample Output Display:

```
A20 gate disabled.
```

2.3.14 DB Command

The DB command allows the developer to set the default memory display format to *bytes*, and then to display memory at the specified address, or at the address immediately following the last byte displayed with the last D, DB, DW, or DD command.

Command Syntax:

`DB` *Address*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode or 0:32 physical address of memory to be displayed in bytes. If not specified, then the address is assumed to be the address immediately following the last byte displayed by the last D, DB, DW, or DD command.

Sample Output Display:

```
0040:0000 f8 03 00 00 00 00 00 00:bc 03 78 03 00 00 00 00  o.......L.x.....
0040:0010 7d 82 00 80 02 00 00 00:00 00 2c 00 2c 00 13 1f  }e.C......,.,...
0040:0020 13 1f 3f 35 0d 1c 03 2e:64 20 0d 1c 6f 18 75 16  ..>5....d...o.u.
0040:0030 74 14 0d 1c 62 30 6c 26:0d 1c 3f 35 0d 1c 01 00  t...b0l&..?5....
0040:0040 24 00 04 00 00 00 01 06:02 07 50 00 00 40 00 00  #.........P..@..
0040:0050 0b 18 00 00 00 00 00 00:00 00 00 00 00 00 00 00  <...............
0040:0060 07 00 00 b4 03 29 30 03:00 00 c8 00 b1 93 01 00  .....)0......o..
0040:0070 00 00 00 00 00 01 81 00:14 14 14 14 01 01 01 01  ......u.........
```

2.3.15 DCACHE Command

The DCACHE command allows the developer disable CPU (L1) and chipset (L2) cache in the system using the methods configured in the BIOS adaptation. This can be used to determine if the cache is working properly.

Command Syntax:

```
DCACHE
```

Parameters:

none.

Sample Output Display:

```
Cache disabled.
```

2.3.16 DD Command

The DD command allows the developer to set the default memory display format to *doublewords*, and then to display memory at the specified address, or at the address immediately following the last byte displayed with the last D, DB, DW, or DD command. Data displayed in this format is in big-endian format (the numbers are real hexadecimal numbers that have been formatted by the debugger by swapping the low and high bytes of each word, and the low and high words of each doubleword.)

Command Syntax:

`DD` *Address*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode or 0:32 physical address of memory to be displayed in doublewords. If not specified, then the address is assumed to be the address immediately following the last byte displayed by the last D, DB, DW, or DD command.

Sample Output Display:

```
0090:0000    6483:b3ea 4300:0004 7279:706f 7468:6769
0090:0010    2943:2820 3839:3120 6547:2039 6172:656e
0090:0020    6f53:206c 6177:7466 2000:6572 2020:2020
0090:0030    2020:2020 2020:2020 2020:2020 2020:2020
0090:0040    454c:4946 4346:0053 4200:5342 4546:4655
0090:0050    4300:5352 544e:554f 4400:5952 434b:5349
0090:0060    4548:4341 4552:4200 5600:4b41 4649:5245
0090:0070    5346:0059 4544:0044 4543:4956 4d4f:4300
```

2.3.17 DW Command

The DW command allows the developer to set the default memory display format to *words*, and then to display memory at the specified address, or at the address immediately following the last byte displayed with the last D, DB, DW, or DD command. Data displayed in this format is in big-endian format (the numbers are real hexadecimal numbers that have been formatted by the debugger by swapping the low and high bytes of each word.)

Command Syntax:

`DW` *Address*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode or 0:32 physical address of memory to be displayed in words. If not specified, then the address is assumed to be the address immediately following the last byte displayed by the last D, DB, DW, or DD command.

Sample Output Display:

```
0090:0000    b3ea 6483 0004 4300 706f 7279 6769 7468
0090:0010    2820 2943 3120 3839 2039 6547 656e 6172
0090:0020    206c 6f53 7466 6177 6572 2000 2020 2020
0090:0030    2020 2020 2020 2020 2020 2020 2020 2020
0090:0040    4946 454c 0053 4346 5342 4200 4655 4546
0090:0050    5352 4300 554f 544e 5952 4400 5349 434b
0090:0060    4341 4548 4200 4552 4b41 5600 5245 4649
0090:0070    0059 5346 0044 4544 4956 4543 4300 4d4f
```

2.3.18 E Command

The E command allows the developer to change a series of consecutive 8-bit storage locations in memory.

Command Syntax:

`E` *Address Value1 [Value2 [Value3...]]*

Parameters:

Address - A required parameter that specifies the 16:16 real-mode or or 0:32 physical address where the first byte in the sequence is to be stored. Subsequent bytes (if specified) are stored in consecutively higher bytes in memory.

Value1, Value2, etc. - A required set of one or more expressions that specify the hexadecimal 8-bit values to be stored at the specified address in memory.

Sample Output Display:

none.

2.3.19 EA20 Command

The EA20 command allows the developer enable the A20 gate using the method configured in the BIOS adaptation.

Command Syntax:

```
EA20
```

Parameters:

none.

Sample Output Display:

```
A20 gate enabled.
```

2.3.20 ECACHE Command

The ECACHE command allows the developer enable CPU (L1) and chipset (L2) cache in the system using the methods configured in the BIOS adaptation. This can be used to determine if the cache is working properly.

Command Syntax:

```
ECACHE
```

Parameters:

none.

Sample Output Display:

```
Cache enabled.
```

2.3.21 EFL Command

The EFL command allows the developer to erase a block of sectored Flash supported by the Flash device driver enabled in the core BIOS, if available.

This command uses the debugger's parsing routines that allow entry of 16:16 (real-mode) addresses, although the address that is actually being entered is a 32-bit physical address. The address is specified in two 16-bit parts, separated by a colon. This address format is purely for convenience and has nothing to do with 16:16 segment:offset addressing.

Command Syntax:

`EFL` *HighPhysAddr*:*LowPhysAddr*

Parameters:

HighPhysAddr - The top 16 bits of a 32-bit physical address that points to the first byte of a Flash block to be erased.

LowPhysAddr - The bottom 16 bits of a 32-bit physical address that points to the first byte of a Flash block to be erased.

Sample Output Display:

```
Flash block erased.
```

2.3.22 G Command

The G command allows the developer to resume execution from within the debugger.

Command Syntax:

`G` *[Address]*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode address of a breakpoint to be set *before* execution begins at the current CS:IP address. If not specified, no breakpoint will be set.

Sample Output Display:

none.

2.3.23 HELP Command

The HELP command allows the developer to display a summary of commands that are supported by the debugger.

Command Syntax:

```
HELP
```

Parameters:

none.

Sample Output Display:

Short summary of available commands.

2.3.24 I Command

The I command allows the developer to issue a read to a byte-wide I/O port in the system. The value read from the port is displayed on the console.

Command Syntax:

`I` *IoAddress*

Parameters:

IoAddress - A required expression that specifies the hexadecimal I/O port to read the 8-bit quantity from.

Sample Output Display:

```
12
```

2.3.25 ID Command

The ID command allows the developer to issue a read to a dword-wide I/O port in the system. The value read from the port is displayed on the console.

Command Syntax:

`ID` *IoAddress*

Parameters:

IoAddress - A required expression that specifies the hexadecimal I/O port to read the 32-bit quantity from.

Sample Output Display:

```
12345678
```

2.3.26 IW Command

The IW command allows the developer to issue a read to a word-wide I/O port in the system. The value read from the port is displayed on the console.

Command Syntax:

`IW` *IoAddress*

Parameters:

IoAddress - A required expression that specifies the hexadecimal I/O port to read the 16-bit quantity from.

Sample Output Display:

```
1234
```

2.3.27 LFL Command

The LFL command allows the developer to lock a block of sectored Flash supported by the Flash device driver enabled in the core BIOS, if available.

This command uses the debugger's parsing routines that allow entry of 16:16 (real-mode) addresses, although the address that is actually being entered is a 32-bit physical address. The address is specified in two 16-bit parts, separated by a colon. This address format is purely for convenience and has nothing to do with 16:16 segment:offset addressing.

Command Syntax:

`LFL` *HighPhysAddr*:*LowPhysAddr*

Parameters:

HighPhysAddr - The top 16 bits of a 32-bit physical address that points to the first byte of a Flash block to be locked.

LowPhysAddr - The bottom 16 bits of a 32-bit physical address that points to the first byte of a Flash block to be locked.

Sample Output Display:

```
Flash block locked.
```

2.3.28 MASK Command

The MASK command allows the developer to specify a bit mask, called the “enabled” bit mask, that is used by Embedded DOS-ROM internal debugging macros (XPRINTF) at run-time, on certain platforms.

XPRINTF statements in the Embedded DOS-ROM kernel specify a bit mask that is ORed with the “enabled” bitmask. If any bits match, then the XPRINTF statement is executed.

This feature allows a developer to add or modify code to the Embedded DOS-ROM kernel and place actual debugging code in the kernel, tied to developer-assigned bits in this bit mask. Then, these bits can be selectively enabled or disabled using the Embedded BIOS debugger with this command.

Command Syntax:

`MASK` *BitMask*

Parameters:

BitMask - A required expression that specifies a 16-bit value containing a bit pattern to be used by the XPRINTF macros in debug-aware Embedded DOS-ROM kernel builds.

Sample Output Display:

None.

2.3.29 MODE Command

The MODE command allows the developer to change the mode of the current video output device. This works by issuing an INT 10h, function 00h, specifying the operand’s value as the video mode.

Most commonly, this feature is used to reset the video mode after some graphics program has run, so that debugger output is visible on the screen. For example, if a graphics program, such as Windows, has painted the screen in some graphics mode, and CTRL-SHIFT has been used to break into the debugger, then the debugger’s output won’t be visible as text, but as a dot spray on the screen. Typing “MODE 7” would cause the debugger to reset the video card (and monitor) to mode 7, which is the standard monochrome mode.

Command Syntax:

`MODE` *VideoMode*

Parameters:

VideoMode - A required expression that specifies a new video mode to set on the current video display.

Sample Output Display:

None.

2.3.30 O Command

The O command allows the developer to issue a write to a byte-wide I/O port in the system. The value written to the port is specified as the second parameter.

Command Syntax:

`O` *IoAddress Value [... Value]*

Parameters:

IoAddress - A required expression that specifies the hexadecimal I/O port to write the 8-bit quantity to.

Value - A required expression that specifies the hexadecimal 8-bit value to write to the I/O port. If more than one *Value* is specified, then each value is written to the I/O port in the specified order, with interrupts disabled and no intervening I/O cycles.

Sample Output Display:

none.

2.3.31 OD Command

The OD command allows the developer to issue a write to a dword-wide I/O port in the system. The value written to the port is specified as the second parameter.

Command Syntax:

`OD` *IoAddress Value [... Value]*

Parameters:

IoAddress - A required expression that specifies the hexadecimal I/O port to write the 32-bit quantity to.

Value - A required expression parameter that specifies the hexadecimal 32-bit value to write to the I/O port. If more than one *Value* is specified, then each value is written to the I/O port in the specified order, with interrupts disabled and no intervening I/O cycles.

Sample Output Display:

none.

2.3.32 OW Command

The OW command allows the developer to issue a write to a word-wide I/O port in the system. The value written to the port is specified as the second parameter.

Command Syntax:

`OW` *IoAddress Value [... Value]*

Parameters:

IoAddress - A required expression that specifies the hexadecimal I/O port to write the 16-bit quantity to.

Value - A required expression that specifies the hexadecimal 16-bit value to write to the I/O port. If more than one *Value* is specified, then each value is written to the I/O port in the specified order, with interrupts disabled and no intervening I/O cycles.

Sample Output Display:

none.

2.3.33 PCIR Command

The PCIR command allows the developer to read a 32-bit doubleword from the PCI configuration space associated with a device and function specified by the user.

Command Syntax:

`PCIR` *Index Device Function*

Parameters:

Index - A required expression that specifies the index into the configuration space where the 32-bit doubleword will be read from.

Device - A required expression parameter that specifies the number of the device from which the data will be read.

Function - A required expression that specifies the device's function number associated with the information to be read.

Sample Output Display:

```
12345678
```

2.3.34 PCIW Command

The PCIW command allows the developer to write a 32-bit doubleword to the PCI configuration space associated with a device and function specified by the user.

Command Syntax:

`PCIR` *Index Data Device Function*

Parameters:

Index - A required expression that specifies the index into the configuration space where the 32-bit doubleword will be written to.

Data - A required expression that specifies the 32-bit hexadecimal data to be written.

Device - A required expression parameter that specifies the number of the device to which the data will be written.

Function - A required expression that specifies the device's function number associated with the information to be written.

Sample Output Display:

none.

2.3.35 R Command

The R command allows the developer to display the contents of the general register set using the display format last commanded with R32 or R16. If this is the first register display command, then the initial register display format is selected based on whether 386 registers are available on the target or not.

Command Syntax:

`R`

Parameters:

none.

Sample Output Display:

```
EMBEDDED BIOS Debugger [IN BIOS] Copyright (C) 1998 General Software
AX=0093  BX=007a  CX=0001  DX=3d26  SI=001e  DI=0000  BP=03b6
CS=f000  DS=0040  ES=157b  SS=157b  SP=037e  IP=ebc3  NV UP EI NG NA PO ZR NC
```

```
f000:ebc3          cli
```

2.3.36 R16 Command

The R16 command allows the developer to display the contents of the general register set using the 16-bit display format.

Command Syntax:

```
        R16
```

Parameters:

none.

Sample Output Display:

```
EMBEDDED BIOS Debugger [IN BIOS] Copyright (C) 1998 General Software
AX=0093  BX=007a  CX=0001  DX=3d26  SI=001e  DI=0000  BP=03b6
CS=f000  DS=0040  ES=157b  SS=157b  SP=037e  IP=ebc3  NV UP EI NG NA PO ZR NC
f000:ebc3          cli
```

2.3.37 R32 Command

The R32 command allows the developer to display the contents of the general register set using the 32-bit display format.

Command Syntax:

```
        R32
```

Parameters:

none.

Sample Output Display:

```
EMBEDDED BIOS Debugger [IN BIOS] Copyright (C) 1998 General Software
EAX = 12345678  CS:EIP = F000:00000149  EFL = 001c213A
EBX = 00000001  SS:ESP = 02C0:00007FFE  EBP = 0000199C
ECX = 179D248E  DS:ESI = 74AB:00000511  FS  = 0000
EDX = 5555AAAA  ES:EDI = F000:0000E000  GS  = 0000
F000:00000149   cli
```

2.3.38 RC Command

The RC command allows the developer to read the contents of battery-backed CMOS memory. Either one byte of CMOS may be displayed, or the entire CMOS contents may be displayed.

Command Syntax:

`RC` [*CmosIndex*]

Parameters:

CmosIndex - An optional expression that specifies the CMOS address (actually, an index into the part) of the CMOS memory to be displayed. If no index is supplied, then the entire contents of CMOS are displayed.

Sample Output Display:

```
Addr  CMOS memory contents...
0000: 00 00 00 00 00 00 01 01 : 01 80 00 00 00 80 00 00
0010: 40 00 00 00 31 80 02 00 : 04 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 f7
0030: 00 00 19 03
```

2.3.39 RD Command

The RD command allows the developer to issue an INT 13h read command so that hard drives, floppy disks, and their emulators, may be tested in the debugger environment. Operands of the RD command specify arguments that are normally passed in registers to INT 13h.

Command Syntax:

`RD` *DriveNo SectorNo HeadNo TrackNo Address*

Parameters:

DriveNo - An 8-bit expression that specifies the INT 13h unit number associated with the device to be read. For example, 0 is the first floppy, 1 is the second floppy, 80 is the first hard drive, and 81 (hexadecimal) is the second hard drive.

SectorNo - An 8-bit expression that specifies the sector number to be read. Sector numbers start with 1 and continue to the last sector number per track. For example, a 1.44MB diskette has sector numbers ranging from 1 to 18 (12 hexadecimal.)

HeadNo - An 8-bit expression that specifies the head number to be read. Head numbers start with 0 and continue to the last head number. For example, a 1.44MB diskette has head numbers ranging from 0 to 1.

TrackNo - A 16-bit expression that specifies the track number to be read. Track numbers start with 0 and continue to the last track per cylinder. For example, a 1.44MB diskette has track numbers ranging from 0 to 79 (4f hexadecimal.)

Address - A 16:16 real-mode address (physical addresses are not permitted) that specifies the memory location where the 512 byte sector will be transferred.

Sample Output Display:

```
Drive 00h, Sector 01h, Head 01h, Track 0042h read, status=00h.
```

2.3.40 REBOOT Command

The REBOOT command allows the developer to reboot the system without removing power to the machine. This command causes the core BIOS to execute the OEM-defined reboot sequence, which may involve only the CPU, port 92h, the Chipset Personality Module, or the CPU Personality Module.

Note that the CPU restart vector on 286 and above processors is *not* F000:FFF0. These CPUs actually execute out of the very top of their physical address space, which in some cases is occupied by a boot loader such as the one provided by CyberQuest. Even though the CPU is executing out of memory above the 1MB address mark, it is still executing in real mode, not protected mode (really, real mode). Once the CS register is reloaded with any value, the CPU disables the upper address lines, and typically, continues to execute at the 8086-compatible reboot address, F000:FFF0.

On 286 and above CPUs, EMBEDDED BIOS enables the A20 line before rebooting the system. This allows special boot loaders to execute.

Command Syntax:

```
REBOOT
```

Parameters:

none.

Sample Output Display:

none.

2.3.41 RFL Command

The RFL command allows the developer to read data from a block of sectored Flash supported by the Flash device driver enabled in the core BIOS, if available. The data are displayed in words, because some Flash arrays only support word accesses.

This command uses the debugger's parsing routines that allow entry of 16:16 (real-mode) addresses, although the address that is actually being entered is a 32-bit physical address. The address is specified in two 16-bit parts, separated by a colon. This address format is purely for convenience and has nothing to do with 16:16 segment:offset addressing.

If the operand is not specified, then reading will continue where the last RFL command left off.

Command Syntax:

`RFL` [*HighPhysAddr*:*LowPhysAddr*]

Parameters:

HighPhysAddr - The top 16 bits of a 32-bit physical address that points to the first word of a Flash block to be read.

LowPhysAddr - The bottom 16 bits of a 32-bit physical address that points to the first word of a Flash block to be read.

Sample Output Display:

```
03a0:0000    b3ea 6483 0004 4300 706f 7279 6769 7468
03a0:0010    2820 2943 3120 3839 2039 6547 656e 6172
03a0:0020    206c 6f53 7466 6177 6572 2000 2020 2020
03a0:0030    2020 2020 2020 2020 2020 2020 2020 2020
03a0:0040    4946 454c 0053 4346 5342 4200 4655 4546
03a0:0050    5352 4300 554f 544e 5952 4400 5349 434b
03a0:0060    4341 4548 4200 4552 4b41 5600 5245 4649
03a0:0070    0059 5346 0044 4544 4956 4543 4300 4d4f
```

2.3.42 SFL Command

The SFL command allows the developer to write a 16-bit pattern to a specified number of words in a Flash array. This is used in situations where a Flash block must be written with all zeroes before erasing it.

Command Syntax:

`SFL` *HighPhysAddr*:*LowPhysAddr Count Word*

Parameters:

HighPhysAddr - The top 16 bits of a 32-bit physical address that points to the first word of a Flash block to be written.

LowPhysAddr - The bottom 16 bits of a 32-bit physical address that points to the first word of a Flash block to be written.

Count - A required expression that specifies the number of words to write in hexadecimal.

Word - A required expression that specifies the 16-bit value to be stored in each word.

Sample Output Display:

```
Data written to Flash.
```

2.3.43 SO Command

The SO command allows the developer to redirect special debugging output from the `XPRINTF` macro in Embedded DOS-ROM to its own output device, such as CON, or COM1-COM4. For more information about `XPRINTF` debugging output see the section on the `MASK` command in this chapter.

Command Syntax:

`SO` *Device*

Parameters:

Device - A required parameter that specifies the new console to redirect Embedded DOS-ROM's `XPRINTF` output to. Supported device names are: CON, COM1, COM2, COM3, and COM4.

Sample Output Display:

none.

2.3.44 T Command

The T command allows the developer to trace through the current instruction and stop execution before the next one is executed. CALL and INT instructions are single-stepped by pushing into the called code; this command does not "step over" the instruction.

Command Syntax:

```
T
```

Parameters:

none.

Sample Output Display:

```
EMBEDDED BIOS Debugger [IN BIOS] Copyright (C) 1998 General Software
AX=0093  BX=007a  CX=0001  DX=3d26  SI=001e  DI=0000  BP=03b6
CS=f000  DS=0040  ES=157b  SS=157b  SP=037e  IP=ebc3  NV UP EI NG NA PO ZR NC
f000:ebc3         cli
```

2.3.45 TIME Command

The TIME command allows the developer to obtain a concrete CPU performance number associated with the target running the BIOS. The TIME command uses its operand value as a number of times to execute a lengthy loop of instructions that perform no useful work other than cause a delay before the prompt comes back.

As the operand's value increases, so does the time it takes for the TIME command to complete and return to the prompt. The relationship between the operand value and the time to complete the command is linear, making it possible to determine how much of a performance improvement certain changes in chipset programming, etc. is incurred.

Here is a simple way of measuring performance improvements:

1. Start with a simple system before the modifications. Suppose, for the sake of argument, that you are interested in how the CPU's incoming clock divisor (manipulated through some chipset register) affects CPU performance. Boot to the debugger, and type "TIME 10". We've chosen 10 here because it is a good starting point. Measure how long this takes.

2. Probably, 10 turned out to be too short or too long. However, you'll definitely know that, because your measurement will be hard to make in the short case, or difficult to wait for, in the

long case. Come up with a better number (*n*) and run TIME on it. Use your stopwatch to measure how much time it takes. For real accuracy, we recommend some interval on the order of 30 seconds or so, to account for delays in starting and stopping the watch. Record the number of seconds it took to perform the TIME command. Call that *x*, here.

3. Now manipulate your chipset registers with the CSR and CSW commands.

4. Run the same TIME command with *n* as its parameter. Record the number of seconds it took to perform this TIME command. Call that *y*, here.

5. Now compute the performance improvement as (*y/x)*.

Command Syntax:

`TIME` *DelayFactor*

Parameters:

DelayFactor - A 16-bit expression specifying the amount of "work" to perform. There are many factors which, combined with this factor, cause the TIME command to delay a certain amount of time. *DelayFactor* is a linear parameter, which means that time taken to perform the command increases linearly with an increase in *DelayFactor* itself.

Sample Output Display:

none.

2.3.46 TORAM Command

The TORAM command copies the BIOS into low memory at **CONFIG_FLASH_CODESEG** and transfers control to the BIOS there. This allows the BIOS to run from RAM during tests involving reconfiguring chipset parameters that relate to the BIOS ROM.

Command Syntax:

```
TORAM
```

Parameters:

none.

Sample Output Display:

none.

2.3.47 U Command

The U command allows the developer to display the contents of memory as a series of consecutive machine instructions. The instructions are formatted as 16-bit or 32-bit, depending on the last unassembly command, whether U, U16, or U32.

By default, the U command unassembles at the current CS:IP address after a debugger break-in. Subsequent U commands display the next few instructions, and so on. Specifying a new address with the U command causes subsequent U commands to display the instructions following the last U command.

Command Syntax:

`U` *[Address]*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode or 0:32 physical address of the first instruction to be decoded and displayed. If not specified, the display will start with the first instruction that follows the one last displayed in a U command.

Sample Output Display:

```
033f:620b         mov     di, [0068]
033f:620f         mov     [di+06], ss
033f:6212         mov     [di+04], sp
033f:6215         mov     [di+02], fffd
033f:6219         call    61b7h
033f:621c         bkpt
033f:621d         retn
033f:621e         push    ds
```

2.3.48 U16 Command

The U16 command allows the developer to display the contents of memory as a series of consecutive machine instructions. The instructions are displayed in 16-bit format (16 bit instruction offsets, etc.)

By default, the U command unassembles at the current CS:IP address after a debugger break-in. Subsequent U commands display the next few instructions, and so on. Specifying a new address with the U command causes subsequent U commands to display the instructions following the last U command.

Command Syntax:

`U16` *[Address]*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode or 0:32 physical address of the first instruction to be decoded and displayed. If not specified, the display will start with the first instruction that follows the one last displayed in a U command.

Sample Output Display:

```
033f:620b          mov     di, [0068]
033f:620f          mov     [di+06], ss
033f:6212          mov     [di+04], sp
033f:6215          mov     [di+02], fffd
033f:6219          call    61b7h
033f:621c          bkpt
033f:621d          retn
033f:621e          push    ds
```

2.3.49 U32 Command

The U32 command allows the developer to display the contents of memory as a series of consecutive machine instructions. The instructions are displayed in 32-bit format (32 bit instruction offsets, etc.)

By default, the U32 command unassembles at the current CS:IP address after a debugger break-in. Subsequent U commands display the next few instructions, and so on. Specifying a new address with the U-type command causes subsequent U commands to display the instructions following the last U command.

Command Syntax:

`U32` *[Address]*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode or 0:32 physical address of the first instruction to be decoded and displayed. If not specified, the display will start with the first instruction that follows the one last displayed in a U command.

Sample Output Display:

```
033f:0000620b          mov     di, [00000068]
033f:0000620f          mov     [di+0006], ss
033f:00006212          mov     [di+0004], sp
033f:00006215          mov     [di+0002], fffffffd
033f:00006219          call    61b7h
033f:0000621c          bkpt
033f:0000621d          retn
033f:0000621e          push    ds
```

2.3.50 UFL Command

The UFL command allows the developer to update an area of Flash from another area of memory (such as the BIOS area at F000:0000).

This command copies the contents of memory specified by the 16:16 real mode address to the physical address.

The Flash must be erased before the update will work, because this command does not automatically erase the Flash before writing to it.

Command Syntax:

`UFL` *HighPhysAddr*:*LowPhysAddr Count SourceAddress*

Parameters:

HighPhysAddr - The top 16 bits of a 32-bit physical address that points to the first word of a Flash block to be written.

LowPhysAddr - The bottom 16 bits of a 32-bit physical address that points to the first word of a Flash block to be written.

Count - A required expression that specifies the number of words to copy in hexadecimal.

SourceAddress - Specifies the 16:16 real-mode address of an area of memory to be copied to the Flash.

Sample Output Display:

```
Flash Updated.
```

2.3.51 V Command

The V command allows the developer to display the contents of an interrupt vector by its number and save the address for a U command so that the code pointed to by that interrupt vector can be disassembled.

The V command is implemented solely to save the OEM time during debugging. The same results can be achieved with the DD command to display the interrupt vector table.

Command Syntax:

`V` *VectorNumber*

Parameters:

VectorNumber - A 16-bit expression that specifies a vector number from 00h to ffh, inclusive.

Sample Output Display:

```
Interrupt Vector 03h Contents:
033f: 620b         mov     di, [00000068]
```

2.3.52 WATCH Command

The WATCH command allows the developer to enable watchpoints inside the core BIOS flagged with **INTENTRY** and **INTEXIT** macro instructions in the source code.

All of the major interrupt service handlers in the BIOS (such as those for INT 10h, INT 11h, and so on) call these macros, one for entry and one for exit. Using the WATCH command, the developer can cause these macros to invoke the debugger's register dump facility to see the general registers on entry and exit to those interrupt handlers. This allows debugging of new BIOS code or analysis of requests made by higher-layer software such as DOS or Windows.

The WATCH command accepts one or more interrupt numbers as operands. If no operands are specified, then the current list of interrupts being watched is displayed. If operands are specified, then their watch status is toggled. So for example, to enable the watchpoint for the INT 10h service, "`WATCH 10`" would be specified. To disable the same watchpoint, the same command would be issued again.

Command Syntax:

`WATCH` *[IntNo [...IntNo]]*

Parameters:

IntNo - A 16-bit expression that specifies a BIOS service interrupt number to watch. Several of these may be specified as arguments.

Sample Output Display:

```
Watchpoint list:  10  11  15  19
```

2.3.53 WC Command

The WC command allows the developer to write a byte to battery-backed CMOS memory at the specified index.

Command Syntax:

`WC` *CmosIndex Value*

Parameters:

CmosIndex - A required expression that specifies the CMOS address (actually, an index into the part) of the CMOS memory to write to.

Value - A required expression that specifies the value to be stored in the specified CMOS location.

Sample Output Display:

none.

2.3.54 WCOMx Command

The WCOMx command allows the developer test a serial port by writing a hexadecimal value to a specified COM port a specified number of times. With large repeat values, the same character can be written out effectively continuously, so that serial ports can be tested with a logic analyzer, remote terminal software, or logic probe.

A period is printed on the primary debugging console to show the progress of writing to the UART, although the actual output goes out the specified device, which is typically not the debug output device.

Command Syntax:

`WCOMx` *ByteToWrite RepeatCount*

Parameters:

x - 1 for COM1, or 2 for COM2.

ByteToWrite - A required 16-bit expression that specifies the value to be written to the output data port of the UART.

RepeatCount - A required 16-bit expression that specifies the number of times to write the value to the UART in succession.

Sample Output Display:

```
Writing to COM1..........
```

2.3.55 WD Command

The WD command allows the developer to issue an INT 13h write command so that hard drives, floppy disks, and their emulators, may be tested in the debugger environment. Operands of the WD command specify arguments that are normally passed in registers to INT 13h.

Command Syntax:

`WD` *DriveNo SectorNo HeadNo TrackNo Address*

Parameters:

DriveNo - An 8-bit expression that specifies the INT 13h unit number associated with the device to be written. For example, 0 is the first floppy, 1 is the second floppy, 80 is the first hard drive, and 81 (hexadecimal) is the second hard drive.

SectorNo - An 8-bit expression that specifies the sector number to be written. Sector numbers start with 1 and continue to the last sector number per track. For example, a 1.44MB diskette has sector numbers ranging from 1 to 18 (12 hexadecimal.)

HeadNo - An 8-bit expression that specifies the head number to be written. Head numbers start with 0 and continue to the last head number. For example, a 1.44MB diskette has head numbers ranging from 0 to 1.

TrackNo - A 16-bit expression that specifies the track number to be written. Track numbers start with 0 and continue to the last track per cylinder. For example, a 1.44MB diskette has track numbers ranging from 0 to 79 (4f hexadecimal.)

Address - A 16:16 real-mode address (physical addresses are not permitted) that specifies the memory location where the 512 byte sector will be copied from.

Sample Output Display:

```
Drive 00h, Sector 01h, Head 01h, Track 0042h written, status=00h.
```

2.3.56 WFL Command

The WFL command allows the developer to write words of data to a block of sectored Flash supported by the Flash device driver enabled in the core BIOS, if available. The data are written in words, because some Flash arrays only support word accesses.

This command uses the debugger's parsing routines that allow entry of 16:16 (real-mode) addresses, although the address that is actually being entered is a 32-bit physical address. The address is specified in two 16-bit parts, separated by a colon. This address format is purely for convenience and has nothing to do with 16:16 segment:offset addressing.

Multiple data words may be specified on the command line, indicating that these words should be written to consecutive word addresses.

Command Syntax:

`WFL` *HighPhysAddr*:*LowPhysAddr Word1* [*Word2*] [*Word3*]...

Parameters:

HighPhysAddr - The top 16 bits of a 32-bit physical address that points to the first word of a Flash block to be written.

LowPhysAddr - The bottom 16 bits of a 32-bit physical address that points to the first word of a Flash block to be written.

Sample Output Display:

```
Data written to Flash.
```

2.3.57 WP Command

The WP command allows the developer to set a data watchpoint on a 16-bit storage area at the specified address. While the watchpoint is set, the processor enters *trace mode*, allowing the debugger to check the status of the storage area after the execution of each instruction to see if it has changed.

While it slows execution considerably (10x or more), a watchpoint can be very useful for finding instructions that are trashing memory.

Command Syntax:

`WP` *[Address]*

Parameters:

Address - An optional parameter that specifies the 16:16 real-mode address of a 16-bit storage location in memory to be monitored. If not specified, the active watchpoint (if any) is cleared.

Sample Output Display:

```
Watchpoint saved.
```

Chapter 3

BIOS FUNCTION REFERENCE

This chapter defines the application programming interface (API) to the BIOS services supported by Embedded BIOS.

It is not intended to document interrupts that strictly do not provide services; consult Chapter 3 for an architectural overview of EMBEDDED BIOS including BIOS up-calls and tables pointed to by interrupt vectors.

3.1 INT 10h, Video BIOS Services

This section explains the video BIOS application program interface (API). The video BIOS is called through software interrupt 10H. Many services are provided to modify or inspect the contents of the video display.

3.1.1 Set Video Mode (00h)

The Set Video Mode video BIOS function is called to set the video mode registers on the video controller for the specified mode of operation. It then clears the screen, positions the cursor at the upper left hand corner of the screen (0,0) and resets the color palette to known values.

Input Parameters:

AH - 00h, indicating the Set Video Mode Function.
AL - Video mode byte.

00h - Text mode, 16 colors, 40x25, 320x200.
01h - Text mode, 16 colors, 40x25, 320x200.
02h - Text mode, 16 colors, 80x25, 640x200.
03h - Text mode, 16 colors, 80x25, 640x200.
04h - Graphics, 4 colors, 40x25, 320x200.
05h - Graphics, 4 colors, 40x25, 320x200.
06h - Graphics, 2 colors, 80x25, 640x200.

07h - Text mode, monochrome, 80x25.
0dh - Graphics, 16 colors, 40x25, 320x200.
0eh - Graphics, 16 colors, 80x25, 640x200.
0fh - Graphics, monochrome, 80x25.
10h - Graphics, 4/16 colors, 80x25, 640x350.

Output Parameters:

AL - Video mode as actually set.

3.1.2 Set Cursor Size (01h)

The Set Cursor Size video BIOS function is called to set the size of the cursor in text modes. The parameters are simply the top and bottom scan lines, in the form of bit masks.

Input Parameters:

AH - 01h, indicating the Set Cursor Size Function.
CH - Top scan line, a complex field as follows:

zz000000b - Must be zero.
00100000b - Shut cursor off.
000xxxxxb - Top scan line.

CL - Bottom scan line, a complex field as follows:

zzz00000b - Must be zero.
000xxxxxb - Bottom scan line.

Output Parameters:

none.

3.1.3 Set Cursor Position (02h)

The Set Cursor Position video BIOS function is called to set the (X,Y) coordinates of the hardware cursor on the screen. The X coordinate is expressed as a column number, beginning with 0 equal to the left-most column on the screen. The Y coordinate is a row number, beginning with 0 equal to the top-most row on the screen.

Input Parameters:

AH - 02h, indicating the Set Cursor Position Function.
BH - Video page number (0 for first page).
DH - Row number (0=top-most row).
DL - Column number (0=left-most column).

Output Parameters:

AX - 0000h.

3.1.4 Read Cursor Position (03h)

The Read Cursor Position video BIOS function is called to return the (X,Y) coordinates of the hardware cursor on the screen. The X coordinate is expressed as a column number, beginning with 0 equal to the left-most column on the screen. The Y coordinate is a row number, beginning with 0 equal to the top-most row on the screen.

Input Parameters:

AH - 03h, indicating the Read Cursor Position Function.
BH - Video page number (0 for first page).

Output Parameters:

AX - 0000h.
CH - Starting cursor scan line.
CL - Ending cursor scan line.
DH - Row number (0=top-most row).
DL - Column number (0=left-most column).

3.1.5 Read Light Pen Position (04h)

The Read Light Pen video BIOS function is called to return the position status of a lightpen. EMBEDDED BIOS does nothing when this function is called.

Input Parameters:

AH - 04h, indicating the Read Light Pen Position Function.

Output Parameters:

AH - Activity flag:

00h - Light pen is not active.
01h - Coordinates returned.

BX - Pixel column (0-319).
CH - Raster line (0-199).
CL - Raster line (0-...).
DH - Row number (0=top-most row).
DL - Column number (0=left-most column).

3.1.6 Select Video Page (05h)

The Select Video Page video BIOS function is called to change the page of the video buffer that is displayed by the 6845 and mapped into its screen regen area (B000H for monochrome, or B800H for color).

Input Parameters:

AH - 05h, indicating the Select Video Page Function.
AL - Page number, where 00h is the first page.

Output Parameters:

none.

3.1.7 Scroll Up Window (06h)

The Scroll Up Window video BIOS function is called to move the contents of a rectangular area on the screen up by a specified number of lines. If the window is specified to cover the entire screen, then the entire screen is scrolled.

Input Parameters:

AH - 06h, indicating the Scroll Up Window Function.
AL - Distance to scroll, in lines (0=blank window).
BH - Attribute byte to use on new lines.
CH - Top row of window.
CL - Left column of window.
DH - Bottom row of window.
DL - Right column of window.

Output Parameters:

none.

3.1.8 Scroll Down Window (07h)

The Scroll Down Window video BIOS function is called to move the contents of a rectangular area on the screen down by a specified number of lines. If the window is specified to cover the entire screen, then the entire screen is scrolled.

Input Parameters:

AH - 07h, indicating the Scroll Down Window Function.
AL - Distance to scroll, in lines (0=blank window).
BH - Attribute byte to use on new lines.
CH - Top row of window.
CL - Left column of window.
DH - Bottom row of window.
DL - Right column of window.

Output Parameters:

none.

3.1.9 Read Char/Attr From Screen (08h)

The Read Char/Attr Pair video BIOS function is called to return the character and attribute located at the current cursor position for the specified page.

Input Parameters:

AH - 08h, indicating the Read Char/Attr Pair Function.
BH - Video page number (0=first page).

Output Parameters:

AH - Attribute byte.
AL - Character.

3.1.10 Write Char/Attr to Screen (09h)

The Write Char/Attr Pair video BIOS function is called to store a character and attribute at the current cursor position for the specified page, without advancing the cursor. The function also allows a repeat count to store multiple characters in sequential columns on the screen.

Input Parameters:

AH - 09h, indicating the Write Char/Attr Pair Function.
AL - Character to store.
BH - Video page number (0=first page).
BL - Attribute byte.
CX - Repeat count.

Output Parameters:

none.

3.1.11 Write Character to Screen (0ah)

The Write Character video BIOS function is called to store a character at the current cursor position for the specified page, without advancing the cursor, using the attribute already defined for that cursor position. The function also allows a repeat count to store multiple characters in sequential columns on the screen.

Input Parameters:

AH - 0ah, indicating the Write Character Function.
AL - Character to store.
BH - Video page number (0=first page).
CX - Repeat count.

Output Parameters:

none.

3.1.12 Set Color Palette (0bh)

The Set Color Palette video BIOS function is called to initialize the 6845's video palette register for graphics modes.

Input Parameters:

AH - 0bh, indicating the Set Color Palette Function.
BH - 00h to set the background color for 320x200 graphics modes, or the border color for 320x200 text modes, or foreground color for 640x200 graphics mode. Otherwise, 01h to set the palette register for 320x200 graphics mode.
BL - Value to set.

Output Parameters:

none.

3.1.13 Write Pixel (0ch)

The Write Pixel video BIOS function is called to store a color value in a pixel addressed by a specified row and column number in graphics mode.

Input Parameters:

AH - 0ch, indicating the Write Pixel Function.
AL - Color to set (set bit 10000000b for XOR mode).
BH - Video page number.
CX - Pixel column number.
DX - Pixel row number.

Output Parameters:

none.

3.1.14 Read Pixel (0dh)

The Read Pixel video BIOS function is called to return a color value in a pixel addressed by a specified row and column number in graphics mode.

Input Parameters:

AH - 0dh, indicating the Read Pixel Function.
BH - Video page number.
CX - Pixel column number.

DX - Pixel row number.

<u>Output Parameters:</u>

AL - Color of pixel.

3.1.15 Write Teletype Mode (0eh)

The Write Teletype video BIOS function is called to write a character to the display at the current cursor location, advancing the cursor to the next column. If the column would extend off the right edge of the screen, the column is reset to 0, and the row is incremented. If the row would move off the end of the screen, then the entire screen is scrolled.

The carriage-return, line-feed, and bell characters perform the same functions that they do on a real teletype.

<u>Input Parameters:</u>

AH - 0eh, indicating the Write Teletype Mode Function.
AL - Character to write.
BL - Foreground color, but only for graphics modes.
BH - Video page number.

<u>Output Parameters:</u>

none.

3.1.16 Return Video Status (0fh)

The Return Video Status video BIOS function is called to return the number of columns on the screen, the current video mode, and the active page number.

<u>Input Parameters:</u>

AH - 0fh, indicating the Return Video Status Function.

<u>Output Parameters:</u>

AH - Columns on the screen.
AL - Current display mode.
BH - Video page number.

3.2 INT 11h, Equipment List Service

The Equipment Status Interrupt is invoked to return the device flags as defined in the BIOS Data Area's **DevFlags** field.

<u>Invocation:</u>

INT 11H

Input Parameters:

none.

Output Parameters:

AX - Device flags.

3.3 INT 12h, Low Memory Size Service

The Low Memory Size Interrupt is invoked to return the size of low memory in kilobytes. This function automatically decrements the returned size by the 1KB Extended BIOS Data Area, located in the top 1KB of low memory.

To determine the amount of available extended memory, use INT 15h function 88h.

Invocation:

INT 12H

Input Parameters:

none.

Output Parameters:

AX - Kilobytes of low memory.

3.4 INT 13h, Disk Services

This section explains the disk BIOS application program interface (API). The disk BIOS is called through software interrupt 13H. Services are provided to reset the disk system, read the status of the last operation, read diskette sectors, write diskette sectors, verify diskette sectors, format disk tracks, read drive parameters, read drive types, detect media changes, set the media type, and set the media type for formatting.

Not all functions are available for all disk types. Note restrictions on each function that apply. For example, the ROM disk does not support write-oriented operations such as Write Sectors, Format Track, and so on.

The following error codes are returned by INT 13h services:

Status	Description
00h	No error
01h	Invalid function
02h	Address mark not found

03h	Media write-protected
04h	Sector not found
05h	Reset failed
06h	Media changed
07h	Hard drive parameter is invalid
08h	DMA overflow occurred
09h	DMA operation crossed 64KB boundary
0ah	Hard drive bad sector
0bh	Hard drive bad track
0ch	Invalid floppy disk media type
0dh	Invalid number of sectors
0eh	Control data address mark found
0fh	DMA arbitration out of range
10h	Unrecoverable read error
11h	Recoverable data area (ECC corrected)
20h	Floppy disk controller failure
40h	Seek to invalid track
80h	Timeout
aah	Hard drive not ready
bbh	Unknown hard disk drive error
cch	Hard drive write error
e0h	Hard drive status register error
ffh	Hard drive sense operation failed

Most disk operations are sector, track, and head-based. When specifying these parameters in INT 13h functions, the sector number is usually specified in the CL CPU register, and the low eight bits of the cylinder number is specified in the CH CPU register. An additional two high bits of cylinder number are stored in the top two bits of the CL CPU register, limiting the sector number stored in the CL CPU register to six bits (values 0-63).

3.4.1 Reset (00h)

The Reset disk BIOS function is called to reset the disk subsystem (ROM, RAM, RFD, floppy, and IDE). This function is used during POST and also whenever an error occurs as a result of a disk operation.

Input Parameters:

AH - 00h, indicating the Reset Function.
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Status code if failure (00h if success).

3.4.2 Read Status (01h)

The Read Status disk BIOS function is called to return the status of the last operation on the specified drive. This status is invalidated when an intervening INT 13h function is invoked (except for this function).

Input Parameters:

AH - 01h, indicating the Read Status Function.
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - 00h.
AL - Disk status code of last operation (00h if success).

3.4.3 Read Sectors (02h)

The Read Sectors disk BIOS function is called to read a sector run from the specified drive into a user-defined buffer. The read must not span a track or head boundary, and the buffer must not cross a 64KB DMA boundary in the physical address space.

Input Parameters:

AH - 02h, indicating the Read Sectors Function.
AL - Number of sectors.
CH - Bottom 8 bits of track number (0-based).
CL - ttssssss, as follows:
    tt = top two bits of 10-bit track number,
    ssssss = 6-bit sector number (1-based).
DH - Head number (0-based).
DL - Drive number.
ES:BX - Address of user buffer.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).
AL - Number of sectors actually read.

3.4.4 Write Sectors (03h)

The Write Sectors disk BIOS function is called to write a sector run to the specified drive from a user-defined buffer. The write must not span a track or head boundary, and the buffer must not cross a 64KB DMA boundary in the physical address space.

This function returns an error when accessing the ROM disk.

Input Parameters:

AH - 03h, indicating the Write Sectors Function.
AL - Number of sectors.
CH - Bottom 8 bits of track number (0-based).
CL - ttssssss, as follows:

tt = top two bits of 10-bit track number,
ssssss = 6-bit sector number (1-based).
DH - Head number (0-based).
DL - Drive number.
ES:BX - Address of user buffer.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).
AL - Number of sectors actually written.

3.4.5 Verify Sectors (04h)

The Verify Sectors disk BIOS function is called to verify that the address marks on a specified track can be read. It does not verify data integrity.

Input Parameters:

AH - 04h, indicating the Verify Sectors Function.
AL - Number of sectors.
CH - Bottom 8 bits of track number (0-based).
CL - ttssssss, as follows:
tt = top two bits of 10-bit track number,
ssssss = 6-bit sector number (1-based).
DH - Head number (0-based).
DL - Drive number.
ES:BX - Address of buffer containing field data.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).

3.4.6 Format Track (05h)

The Format Track disk BIOS function is called to format the specified track of a drive with specific address marks.

This function returns an error when accessing the ROM disk.

Input Parameters:

AH - 05h, indicating the Format Sectors Function.
AL - Number of sectors/this track.
CH - Bottom 8 bits of track number (0-based).
CL - ttssssss, as follows:
tt = top two bits of 10-bit track number,
ssssss = 6-bit sector number (1-based).
DH - Head number (0-based).

DL - Drive number.
ES:BX - Address of buffer containing field data.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).

3.4.7 Read Drive Parameters (08h)

The Read Drive Parameters disk BIOS function is called to return the geometry and disk type information for the specified drive. Additionally, the number of drives like the one specified is returned; i.e., if a floppy drive number is supplied, then the number of floppy drives is returned in the DL CPU register, and so on for hard drives. The ROM, RAM, and RFD drives count as floppy drives.

Input Parameters:

AH - 08h, indicating the Read Drive Parameters Function.
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).
BH - 00h (floppy drives only).
BL - Drive type, as follows (floppy drives only):

01h - 5.25", 360KB, 40 tracks.
02h - 5.25", 1.2MB, 80 tracks.
03h - 3.5", 720KB, 80 tracks.
04h - 3.5", 1.44MB, 80 tracks.

CH - Bottom 8 bits of maximum track number.
CL - ttssssss, as follows:
tt = top two bits of 10-bit maximum track number,
ssssss = 6-bit maximum sector number.

DH - Maximum head number.
DL - Number of drives installed.
ES:DI - Pointer to the diskette parameter table entry for a floppy drive.

3.4.8 Initialize Hard Disk Controller (09h)

The Initialize Hard Disk Controller disk BIOS function is called to initialize the disk controller with the values in the BIOS hard disk parameter tables pointed to by IVT entries 41h and 46h. See Chapter 3 for a description of the data structures pointed to by these tables.

This function returns an error when accessing a floppy disk or its emulator.

Input Parameters:

AH - 09h, indicating the Initialize Hard Disk Controller Function.
DL - Drive number (80h=C:, 81h=D:).

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).

3.4.9 Read Long Sectors (0ah)

The Read Long Sectors disk BIOS function is called to read a sector run from the specified drive into a user-defined buffer with a 4-byte error correction code (ECC) for each sector. The read must not span a track or head boundary, and the buffer must not cross a 64KB DMA boundary in the physical address space.

This function is only valid for hard disk drives.

Input Parameters:

AH - 0ah, indicating the Read Long Sectors Function.
AL - Number of sectors.
CH - Bottom 8 bits of track number (0-based).
CL - ttssssss, as follows:
    tt = top two bits of 10-bit track number,
    ssssss = 6-bit sector number (1-based).
DH - Head number (0-based).
DL - Drive number.
ES:BX - Address of user buffer.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).
AL - Number of sectors actually read.

3.4.10 Write Long Sectors (0bh)

The Write Long Sectors disk BIOS function is called to write a sector run from the specified drive from a user-defined buffer with a 4-byte error correction code (ECC) for each sector. The write must not span a track or head boundary, and the buffer must not cross a 64KB DMA boundary in the physical address space.

This function is only valid for hard disk drives.

Input Parameters:

AH - 0bh, indicating the Write Long Sectors Function.
AL - Number of sectors.

CH - Bottom 8 bits of track number (0-based).
CL - ttssssss, as follows:
tt = top two bits of 10-bit track number,
ssssss = 6-bit sector number (1-based).
DH - Head number (0-based).
DL - Drive number.
ES:BX - Address of user buffer.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).
AL - Number of sectors actually written.

3.4.11 Seek to Cylinder (0ch)

The Seek to Cylinder disk BIOS function is called to position a read/write head on a hard drive over a specified track. No data is transferred during this request.

This function is only valid for hard disk drives.

Input Parameters:

AH - 0ch, indicating the Seek to Cylinder Function.
CH - Bottom 8 bits of track number (0-based).
CL - tt000000, as follows:
tt = top two bits of 10-bit track number.
DH - Head number (0-based).
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).

3.4.12 Reset Hard Disk Controller (0dh)

The Reset Hard Disk Controller disk BIOS function is called to initialize the IDE controller. While function 00h resets both the floppy and hard disk controllers, this function only resets the hard drive controller.

This function is only valid for hard disk drives.

Input Parameters:

AH - 0dh, indicating the Reset Hard Disk Controller Function.
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).

3.4.13 Test Drive Ready (10h)

The Test Drive Ready disk BIOS function is called to verify that the hard drive specified in the DL CPU register is ready to perform additional functions.

This function is only valid for hard disk drives.

Input Parameters:

AH - 10h, indicating the Test Drive Ready Function.
DL - Drive number.

Output Parameters:

CY - Set if failure (not ready), else clear if success (ready).
AH - Disk status code (00h if success).

3.4.14 Recalibrate Drive (11h)

The Recalibrate Drive disk BIOS function is called to recalibrate a hard drive. Externally, this involves restoring all of the read/write heads to the track 0 position. Internally, this also involves rezeroing the feedback loop that determines the head position inside the drive. If read and write requests start encountering frequent correctable errors, this function should be called to recalibrate the heads.

This function is only valid for hard disk drives.

Input Parameters:

AH - 11h, indicating the Recalibrate Drive Function.
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).

3.4.15 Controller Diagnostic (14h)

The Controller Diagnostic disk BIOS function is called to initiate a diagnostic routine on the hard disk controller. The outcome of this diagnostic is returned in the status code.

This function is only valid for hard disk drives.

Input Parameters:

AH - 14h, indicating the Controller Diagnostic Function.
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).

3.4.16 Read Drive Type (15h)

The Read Drive Type disk BIOS function is called to return the disk type information for a disk unit.

Input Parameters:

AH - 15h, indicating the Read Drive Type Function.
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Drive type, as follows:

00h - Drive number is invalid.
01h - Diskette drive with no change line.
02h - Diskette drive with a change line.
03h - Fixed disk.

CX:DX - for fixed disks, a 32-bit number of 512-byte sectors.

3.4.17 Detect Media Change (16h)

The Detect Media Change disk BIOS function is called to return the status of the disk change line for a disk unit.

Input Parameters:

AH - 16h, indicating the Detect Media Change Function.
DL - Drive number.

Output Parameters:

CY - Set if failure, else clear if success.
AH - Change information, as follows:

00h - Diskette change line signal not active.
01h - Invalid drive number.
06h - Change may have occurred.
80h - Drive not ready, or invalid drive.

3.4.18 Set Diskette Type (17h)

The Set Diskette Type disk BIOS function is called to set the data transfer rate for the specified drive.

This function returns an error when accessing a fixed disk.

<u>Input Parameters:</u>

AH - 17h, indicating the Set Diskette Type Function.
AL - Diskette type, as follows:

00h - Reserved.
01h - 360KB diskette in 360KB drive.
02h - 360KB diskette in 1.2MB drive.
03h - 1.2MB diskette in 1.2MB drive.
04h - 720KB diskette in 720KB drive.

DL - Drive number.

<u>Output Parameters:</u>

CY - Set if failure, else clear if success.
AH - Disk status code (00h if success).

3.4.19 Set Media Type for Format (18h)

The Set Media Type for Format disk BIOS function is called to set the media type for the specified drive in order for a FORMAT operation to proceed.

This function returns an error when accessing a fixed disk.

<u>Input Parameters:</u>

AH - 18h, indicating the Set Media Type Function.
CH - Maximum track number (0-based).
CL - Maximum sectors per track (0-based).
DL - Drive number.

<u>Output Parameters:</u>

CY - Set if failure, else clear if success.
AH - Status code, as follows:

00h - Track/sector type supported.
0ch - Media type unknown.
80h - No diskette in drive.
xxh - Disk status code.

ES:DI - Address of diskette parameter table for specified track/sector combination.

3.5 INT 14h, Serial I/O Services

This section explains the serial BIOS application program interface (API). The serial BIOS is called through software interrupt 14H. Services are provided to initialize the serial ports, send characters, receive characters, and read the serial port status.

3.5.1 Initialize Serial Port (00h)

The Initialize Serial Port serial BIOS function is called to initialize the communications parameters for a specific serial port. For greater control over serial port initialization, use the extended serial port initialization function (04h).

Input Parameters:

AH - 00h, indicating the Initialize Serial Port Function.
AL - Serial port initialization parameters:

bbb00000b - Baud rate, as follows:

000b - 110 baud.
001b - 150 baud.
010b - 300 baud.
011b - 600 baud.
100b - 1200 baud.
101b - 2400 baud.
110b - 4800 baud.
111b - 9600 baud.

000pp000b - Parity, as follows:

00b - No parity.
01b - Odd parity.
10b - No parity.
11b - Even parity.

00000s00b - Stop bits, as follows:

0b - One stop bit.
1b - Two stop bits.

00000011b - Data bits, as follows:

10b - 7 data bits.
11b - 8 data bits.

DX - Serial port number (0=COM1, 1=COM2, 2=COM3, 3=COM4).

Output Parameters:

AH - Line status register, as follows:

10000000b - Timeout error occurred.
01000000b - Transmitter shift & holding register empty.
00100000b - Transmitter holding register empty.
00010000b - Break interrupt occurred.
00001000b - Framing error occurred.
00000100b - Parity error occurred.
00000010b - Data overrun error occurred.
00000001b - Data ready.

AL - Modem status register, as follows:

10000000b - Data carrier detect.
01000000b - Ring indicator.
00100000b - Data set ready.
00010000b - Clear to send.
00001000b - Delta data carrier select.
00000100b - Trailing edge ring indicator.
00000010b - Delta data set ready.
00000001b - Delta clear to send.

3.5.2 Send Character (01h)

The Send Character serial BIOS function is called to send a byte over the specified serial communications channel.

Input Parameters:

AH - 01h, indicating the Send Character Function.
AL - Character to send.
DX - Serial port number (0=COM1, 1=COM2, 2=COM3, 3=COM4).

Output Parameters:

AH - Line status register, as follows:

10000000b - Timeout error occurred.
01000000b - Transmitter shift & holding register empty.
00100000b - Transmitter holding register empty.
00010000b - Break interrupt occurred.
00001000b - Framing error occurred.
00000100b - Parity error occurred.
00000010b - Data overrun error occurred.
00000001b - Data ready.

AL - Character sent.

3.5.3 Receive Character (02h)

The Receive Character serial BIOS function is called to receive a byte over the specified serial communications channel.

Input Parameters:

AH - 02h, indicating the Receive Character Function.
DX - Serial port number (0=COM1, 1=COM2, 2=COM3, 3=COM4).

Output Parameters:

AH - Line status register, as follows:

10000000b	- Timeout error occurred.
01000000b	- Transmitter shift & holding register empty.
00100000b	- Transmitter holding register empty.
00010000b	- Break interrupt occurred.
00001000b	- Framing error occurred.
00000100b	- Parity error occurred.
00000010b	- Data overrun error occurred.
00000001b	- Data ready.

AL - Character received.

3.5.4 Read Serial Port Status (03h)

The Read Serial Port Status serial BIOS function is called to read the modem status register and the line status register for the specified serial port.

Input Parameters:

AH - 03h, indicating the Read Serial Port Status Function.
DX - Serial port number (0=COM1, 1=COM2, 2=COM3, 3=COM4).

Output Parameters:

AH - Line status register, as follows:

10000000b	- Timeout error occurred.
01000000b	- Transmitter shift & holding register empty.
00100000b	- Transmitter holding register empty.
00010000b	- Break interrupt occurred.
00001000b	- Framing error occurred.
00000100b	- Parity error occurred.
00000010b	- Data overrun error occurred.
00000001b	- Data ready.

AL - Modem status register, as follows:

10000000b - Data carrier detect.
01000000b - Ring indicator.
00100000b - Data set ready.
00010000b - Clear to send.
00001000b - Delta data carrier select.
00000100b - Trailing edge ring indicator.
00000010b - Delta data set ready.
00000001b - Delta clear to send.

3.5.5 Extended Initialize Serial Port (04h)

The Extended Initialize Serial Port serial BIOS function is called to initialize the communications parameters for a specific serial port, with more architectural room for handling faster ports, up to 115 kbaud.

Note that not all serial ports can be programmed to accommodate all baud rates or protocols. See the CPU Personality Module for your target's CPU class to determine if there are restrictions when initializing on-board CPU serial ports.

Input Parameters:

AH - 04h, indicating the Extended Initialize Serial Port Function.
AL - 00h if no break signal, 01h if break signal.
BH - Parity, as follows:

00h - no parity.
01h - odd parity.
02h - even parity.
03h - stick parity odd.
04h - stick parity even.

BL - Stop bits, as follows:

00h - 1 stop bit.
01h - 2 stop bits if data length is 6, 7, or 8 bits.
02h - 1.5 stop bits if data length is 5 bits.

CH - Data length, as follows:

00h - 5 bits.
01h - 6 bits.
02h - 7 bits.
03h - 8 bits.

CL - Baud rate, as follows:

00h - 110 baud.
01h - 150 baud.
02h - 300 baud.
03h - 600 baud.

04h - 1200 baud.
05h - 2400 baud.
06h - 4800 baud.
07h - 9600 baud.
08h - 19.2 kbaud.
09h - 38.4 kbaud.
0ah - 56 kbaud.
0bh - 115 kbaud.

DX - Serial port number (0=COM1, 1=COM2, 2=COM3, 3=COM4).

Output Parameters:

AH - Line status register, as follows:

10000000b - Timeout error occurred.
01000000b - Transmitter shift & holding register empty.
00100000b - Transmitter holding register empty.
00010000b - Break interrupt occurred.
00001000b - Framing error occurred.
00000100b - Parity error occurred.
00000010b - Data overrun error occurred.
00000001b - Data ready.

AL - Modem status register, as follows:

10000000b - Data carrier detect.
01000000b - Ring indicator.
00100000b - Data set ready.
00010000b - Clear to send.
00001000b - Delta data carrier select.
00000100b - Trailing edge ring indicator.
00000010b - Delta data set ready.
00000001b - Delta clear to send.

3.6 INT 15h, General Services

This section explains the general services BIOS application program interface (API). The general services BIOS is called through software interrupt 15H. Services are provided to support multitasking for device waits, protected mode functions, access to system configuration information, and access the Advanced Power Management services.

Additional services are supported to handle CMOS RAM reading and writing, setting the BIOS **CurrIo** variable to affect console redirection, Flash programming, and returning the EMBEDDED BIOS version number.

3.6.1 Query Port 92h A20 Gate Capability (24h)

The Query Port 92h A20 Gate Capability BIOS function provides information to the caller about whether the application or operating system can switch the A20 gate with port 92h. This is a legacy function used by HIMEM.SYS.

Input Parameters:

AH - 24h, indicating Query Port 92h A20 Gate Capability Function.
AL - subfunction, as follows:
    01h - Enable A20 gate.
    02h - Disable A20 gate.
    03h - Determine if port 92h support is available.

Output Parameters:

CY - set if failure (no port 92h support), else clear if success.
AH - if failure, 86h.

BX - if subfunction 03h, returns the value 2, indicating support available.

3.6.2 Keyboard Intercept Up-Call (4fh)

The Keyboard Intercept system services BIOS up-call is called by the keyboard BIOS interrupt service routine to allow the operating system or application (client) to receive notice of incoming scan codes (both make and break). If no client is available, then the default handler for this routine returns with the CY flag set.

If the client desires to process the incoming scan code, then it must clear the CY flag after processing the data passed in the AL CPU register. This causes the keyboard BIOS to abort further processing of the scan code other than issuing an EOI to the interrupt controller.

If the client does not wish to process the incoming scan code, then it must set the CY flag before returning. At the client's option, it may elect to modify the scan code passed in the AL CPU register so that the keyboard BIOS processes the input differently. The client is cautioned that this technique can lead to keyboard BIOS failure if non-scan codes are processed; other data, such as status codes, are also passed to this routine.

Input Parameters:

AH - 4fh, indicating the Keyboard Intercept Up-Call.
AL - scan code.

Output Parameters:

CY - set if keyboard BIOS should process scan code in AL, else clear if keyboard BIOS should discard the scan code.
AL - scan code as updated by client.

3.6.3 APM Installation Check (5300h)

The Advanced Power Management Installation Check BIOS function is called to determine if Advanced Power Management services are enabled, and if so, which version of the specification it supports.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 00h, indicating Installation Check Subfunction.
BX - 0000h, indicating system BIOS.

Output Parameters:

CY - set if failure, else clear if success.
AH - if failure, 86h.
AH - if success, major version number in BCD.
AL - minor version number in BCD.
BH - ASCII "P" character.
BL - ASCII "M" character.
CX - capabilities flags, as follows:

bit 0 = 1 if 16-bit protected mode interface supported.
bit 1 = 1 if 32-bit protected mode interface supported.
bit 2 = 1 if CPU Idle call slows processor clock speed.
bit 3 = 1 if BIOS Power Management is disabled.

3.6.4 APM Interface Connect (5301h)

The Advanced Power Management Interface Connect BIOS function is called to establish the cooperative interface between the caller and the system BIOS. Before the interface is established, the system BIOS will provide its own power management functionality as implemented by the OEM. Once the interface is established (connected), the system BIOS and the caller will coordinate power management activities together.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 01h, indicating Interface Connect Subfunction.
BX - 0000h, indicating system BIOS.

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

02h - interface connection already in effect.
09h - unrecognized device ID.
86h - APM not supported.

3.6.5 APM Protected Mode 16-Bit Interface Connect (5302h)

The Advanced Power Management Protected Mode 16-Bit Interface Connect BIOS function is called to initialize an optional 16-bit protected mode interface between the caller and the system BIOS. This interface allows a protected mode caller to invoke the system BIOS functions without the need to first switch into real or virtual-86 mode. A caller that does not operate in protected mode may not need to use this call. This function establishes a 16-bit protected mode interface, but this function must be invoked in either real or virtual-86 mode using the INT 15h interface.

<u>Input Parameters:</u>

AH - 53h, indicating an Advanced Power Management Function.
AL - 02h, indicating 16-Bit P/M Interface Connect Subfunction.
BX - 0000h, indicating system BIOS.

<u>Output Parameters:</u>

CY - set if failure, else clear if success.
AH - error code, as follows:

02h - interface connection already in effect.
05h - 16-bit protected mode interface already established.
06h - 16-bit protected mode interface not supported.
09h - unrecognized device ID.
86h - APM not supported.

AX:BX - if success, 16:16 protected mode entrypoint supporting APM requests for the system BIOS.
CX - if success, 16-bit data selector used by APM entrypoint.

3.6.6 APM Protected Mode 32-Bit Interface Connect (5303h)

The Advanced Power Management Protected Mode 32-Bit Interface Connect BIOS function is called to initialize an optional 32-bit protected mode interface between the caller and the system BIOS. This interface allows a protected mode caller to invoke the system BIOS functions without the need to first switch into real or virtual-86 mode. A caller that does not operate in protected mode may not need to use this call. This function establishes a 32-bit protected mode interface, but this function must be invoked in either real or virtual-86 mode using the INT 15h interface.

<u>Input Parameters:</u>

AH - 53h, indicating an Advanced Power Management Function.
AL - 03h, indicating 32-Bit P/M Interface Connect Subfunction.
BX - 0000h, indicating system BIOS.

<u>Output Parameters:</u>

CY - set if failure, else clear if success.
AH - error code, as follows:

02h - interface connection already in effect.
07h - 32-bit protected mode interface already established.
08h - 32-bit protected mode interface not supported.
09h - unrecognized device ID.

86h - APM not supported.

AX:EBX - if success, 16:32 protected mode entrypoint supporting APM requests for the system BIOS.
CX - if success, 16-bit code segment selector used by APM entrypoint.
DX - if success, 16-bit data selector used by APM entrypoint.

3.6.7 APM Interface Disconnect (5304h)

The Advanced Power Management Interface Disconnect BIOS function is called to break the cooperative interaction between the system BIOS and the caller, and in the process restores the system BIOS default functionality. Any protected mode connection set-up by the protected mode interface connection functions are invalidated by this call.

Even though this call returns control of power management to the system BIOS, the parameter values (timer values, enable/disable settings, etc.) in effect at the time of the disconnect will remain in effect.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 04h, indicating Interface Disconnect Subfunction.
BX - 0000h, indicating system BIOS.

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

03h - interface not connected.
09h - unrecognized device ID.
86h - APM not supported.

3.6.8 APM CPU Idle (5305h)

The Advanced Power Management CPU Idle BIOS function is called to inform the system BIOS that the system is currently idle, and that processing should be suspended until the next system event (typically an interrupt) occurs. This function allows the system BIOS to take some implementation specific power saving action, such as a CPU HLT instruction or stopping the CPU clock.

In cases where an interrupt causes the system to leave the idle state, the interrupt may or may not have been serviced when the BIOS returns from the CPU Idle request.

if interrupts are serviced from within the CPU Idle function, the interrupt handler must return to the BIOS when the interrupt processing is completed. The caller cannot use its knowledge of being in the idle state to retain control from an interrupt handler. For example, some system implementations may slow the processor CPU clock rate before waiting on an interrupt, and restore the normal clock rate after the interrupt is serviced but before returning from the idle call.

When the caller regains control from the system BIOS idle routine, it should determine if there is actually any processing to be performed, and reissue the CPU idle call if not. If the caller is a multitasking supervisor, it may be necessary for it to dispatch its applications, allowing them to check for activity that they should then perform.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 05h, indicating CPU Idle Subfunction.

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

86h - APM not supported.

3.6.9 APM CPU Busy (5306h)

The Advanced Power Management CPU Busy BIOS function is called to inform the system BIOS that the system is now busy and processing should continue at full speed. Some system implementations may only be able to slow the CPU clock rate and return in response to the CPU Idle request (see function 5305h). It is expected that the system BIOS will restore the CPU clock reate to its normal rate when it recognizes increased system activity (typically interrupt-driven), but it may be unable to do so when interrupts are hooked by external software that does not invoke BIOS routines.

In cases where this is possible, the caller can ensure the system is running at full speed by invoking the CPU Busy function. Upon return from the APM Installation Check call, bit 2 of the CX CPU register indicates that the system BIOS slows the CPU clock rate during the CPU Idle call. The caller can use this bit to determine if it wishes to call CPU Busy before executing code that it wants to run at full speed.

Calling CPU Busy when the systme is already operating at full speed is discouraged due to the unnecessary call overhead, but the operation is allowed and iwll have no unexpected side effects.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 06h, indicating CPU Busy Subfunction.

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

86h - APM not supported.

3.6.10 APM Set Power State (5307h)

The Advanced Power Management Set Power State BIOS function is called to place the system in the requested state. The system BIOS only responds to power device ID = 0001h (system BIOS).

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 07h, indicating Set Power State Subfunction.
BX - 0001h, indicating system BIOS.
CX - System State ID, as follows:

0000h - Ready (not supported for device ID 0001h).
0001h - Standby.
0002h - Suspend.
0003h - Off (not supported for device ID 0001h).

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

01h - power management functionality disabled.
09h - unrecognized device ID.
0ah - parameter valud in CX out of range.
60h - cannot enter requested state.
86h - APM not supported.

3.6.11 APM Enable/Disable APM Functionality (5308h)

The Advanced Power Management Enable/Disable APM Functionality BIOS function is called to enable or disable all APM automatic power down functionality. When disabled, the system BIOS will not automatically power down devices, enter the standby state, enter the suspended state, or take power saving steps in response to CPU Idle calls. In addition, many system BIOS functions will be disabled and will return error 01h, power management functionality disabled.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 08h, indicating Enable/Disable APM Functionality Subfunction.
BX - ffffh, "enable/disable all power management".
CX - 0 to disable, 1 to enable.

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

01h - power management functionality disabled.
09h - unrecognized device ID.
0ah - parameter valud in CX out of range.
86h - APM not supported.

3.6.12 APM Restore APM Power-On Defaults (5309h)

The Advanced Power Management Restore APM Power-On Defaults BIOS function is called to instruct the BIOS to reinitialize all of its power-on APM defaults.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 09h, indicating Restore APM Power-On Defaults Subfunction.
BX - ffffh, "all power management".

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

09h - unrecognized device ID.
86h - APM not supported.

3.6.13 APM Get Power Status (530ah)

The Advanced Power Management Get Power Status BIOS function is called to return the system BIOS's current power management status.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 0ah, indicating Get Power Status Subfunction.
BX - 0001h.

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

09h - unrecognized device ID.
86h - APM not supported.

BH - if success, A/C line status as follows:

00h - off-line.
01h - on-line.
ffh - unknown.

BL - if success, battery status as follows:

00h - high.
01h - low.

02h - critical.
03h - charging.
ffh - unknown.

CL - if success, remaining battery life, as follows:

0 - 100% = # of full charge
ffh - unknown.

3.6.14 APM Get APM Event (530bh)

The Advanced Power Management Get APM Event BIOS function is called to return the next pending PM event, or indicates if no PM events are pending.

Input Parameters:

AH - 53h, indicating an Advanced Power Management Function.
AL - 0bh, indicating Get APM Event Subfunction.

Output Parameters:

CY - set if failure, else clear if success.
AH - error code, as follows:

03h - interface connection not established.
86h - APM not supported.

BX - if success, PM event code, as follows:

01h - system standby request notification
02h - system suspend request notification
03h - normal resume system notification
04h - critical resume system notification
05h - battery low notification

3.6.15 System Request Key (58h)

The System Request Key BIOS Up-Call is called by the keyboard BIOS interrupt service routine to allow the operating system or application (client) to receive notice that the SysReq key has been pressed or released.

If no client intercepts the up-call, then it will be handled by the default system BIOS handler, which clears the CY flag and sets the AH CPU register to 00h.

Input Parameters:

AH - 85h, indicating a System Request Key Up-Call.
AL - 00h if key pressed, else 01h if key released.

Output Parameters:

CY - clear.
AH - 00h.

3.6.16 Wait Function (86h)

The Wait Function BIOS function is called to delay for a specified number of microseconds so that applications can perform fine timing.

This function must be carefully tuned by the OEM during the adaptation process; it should not be assumed to be accurate until the OEM has done this tuning. The tuning is handled in BPM routine **BoardDelayUsec**.

Input Parameters:

AH - 86h, indicating the Wait Function.
CX:DX - 32-bit number of microseconds to wait.

Output Parameters:

CY - set if failure, else clear if success.

3.6.17 Move Extended Memory Block (87h)

The Move Extended Memory Block BIOS function is called to perform a memory copy using the protected mode capabilities of targets that support protected mode operation.

The memory transfer is specified in the form of a GDT whose real-mode address is passed to the function. Also passed to the function is a number of 16-bit words to copy (beware, this function cannot transfer an odd number of bytes).

Input Parameters:

AH - 87h, indicating the Move Extended Memory Block Function.
CX - number of 16-bit words to copy.
ES:SI - 16:16 real-mode address of GDT describing source and destination addresses for the copy process, formatted as follows:

GDT entry #0 - dummy entry, should be all zeroes.
GDT entry #1 - pointer to GDT.
GDT entry #2 - source buffer.
GDT entry #3 - destination buffer.
GDT entry #4 - reserved by BIOS, do not initialize.
GDT entry #5 - reserved by BIOS, do not initialize.

The format of a GDT and its entries is specified in Intel documentation and is beyond the scope of this manual.

Output Parameters:

CY - set if failure, else clear if success.
AH - status code, as follows:

00h - no error.
01h - RAM parity error occurred during copy.
02h - CPU exception occurred during copy.
03h - gate A20 operation failed.
86h - protected mode services not available.

3.6.18 Extended Memory Size (88h)

The Extended Memory Size BIOS function is called to return the amount of extended memory (that RAM available to the application above the 1MB physical address boundary).

Input Parameters:

AH - 88h, indicating the Extended Memory Size Function.

Output Parameters:

CY - set if failure, else clear if success.
AH - status code, as follows:

86h - protected mode services not available.

AX - if success, extended memory size in 1KB units.

3.6.19 Switch To Protected Mode (89h)

The Switch To Protected Mode BIOS function is called to switch the mode of the processor and establish a GDT for addressability for the remainder of the system's operation.

Input Parameters:

AH - 89h, indicating the Switch To Protected Mode Function.
BH - index into the IDT specifying the base of the first 8 hardware interrupts.
BL - index into the IDT specifying the base of the second 8 hardware interrupts.
ES:SI - 16:16 real-mode address of GDT built by caller, as follows:

GDT entry #0 - dummy entry, should be all zeroes.
GDT entry #1 - pointer to GDT.
GDT entry #2 - pointer to IDT.
GDT entry #3 - pointer to data segment.
GDT entry #4 - pointer to extra segment.
GDT entry #5 - pointer to stack segment.
GDT entry #6 - pointer to code segment.
GDT entry #7 - additional descriptor for BIOS scratch.

The format of a GDT and its entries is specified in Intel documentation and is beyond the scope of this manual.

Output Parameters:

CY - set if failure, else clear if success.
AH - status code, as follows:

00h - no error.
01h - RAM parity error occurred during copy.
02h - CPU exception occurred during copy.
03h - gate A20 operation failed.
86h - protected mode services not available.

3.6.20 Device Busy Up-Call (90h)

The Device Busy BIOS up-call is called by various device management modules within the system BIOS to allow the operating system or application (client) to receive notice of an impending spin-loop within the BIOS to wait for a device to perform a mechanical function. If no client is available, then the default handler for this routine returns with the CY flag set.

If a client is available, it can decide to perform other activities and return when the Device Interrupt up-call is received. When it takes this option, it returns from the Device Busy BIOS up-call with the CY flag cleared. When the BIOS detects that the CY flag is cleared, it does not enter the anticipated spinloop, but instead assumes that the operation has completed or has timed-out.

If the client decides to ignore the Devicy Busy notification and let the BIOS enter its spin-loop, then it returns from the Device Busy BIOS up-call with the CY flag set. This causes the BIOS to continue as though the client had never hooked the INT 15h service in the first place.

Input Parameters:

AH - 90h, indicating a Device Busy Up-Call.
AL - Device type code, as follows:

00h - hard disk drive.
01h - floppy disk drive.
02h - keyboard.
03h - PS/2 mouse.
80h - network.
fch - hard disk reset.
fdh - floppy disk drive motor.
feh - printer.

ES:BX - if AL=80h-ffh, then this register pair points to a request block that is device-dependent.

Output Parameters:

CY - set if caller should enter spinloop, else clear if caller should avoid spinloop and assume device operation has completed or has timed-out.

3.6.21 Device Interrupt Up-Call (91h)

The Device Interrupt BIOS up-call is called by various device management modules within the system BIOS to allow the operating system or application (client) to receive notice of a peripheral device's completion of some event, such as a seek of a disk drive. This allows the client to return control to the BIOS if it transferred control to another task in response to the Device Busy up-call associated with this Device Interrupt up-call. If no client is available, then the default handler for this routine returns with the CY flag set.

If a client is available, it can decide to reschedule the task that was blocked waiting for the completion of the device operation associated with the previous Device Busy up-call. Regardless of the client's decision to perform this action, the BIOS will continue execution of its code upon return from this function without inspecting the CY flag.

Input Parameters:

AH - 91h, indicating a Device Interrupt Up-Call.
AL - Device type code, as follows:

00h - hard disk drive.
01h - floppy disk drive.
02h - keyboard.
03h - PS/2 mouse.
80h - network.
fch - hard disk reset.
fdh - floppy disk drive motor.
feh - printer.

ES:BX - if AL=80h-ffh, then this register pair points to a request block that is device-dependent.

Output Parameters:

none.

3.6.22 Read/Write CMOS RAM Cell (A0h)

The Read/Write CMOS RAM Cell BIOS function is called by application software to access CMOS RAM cells in a hardware-independent manner. This is useful when the application must run on hardware that may not use the ISA-standard ports 70h and 71h for accessing this hardware.

Input Parameters:

AH - A0h, indicating the Read/Write CMOS RAM Cell Function.
AL - 00h for a read operation, or 01h for a write operation.
BL - specifies the CMOS RAM index to read or write.

BH - for writes only, specifies the value to be written.

Output Parameters:

CY - clear if success, else set if failure.
AL - for reads only, contains the value that was read.
AH - status code, as follows:

00h - no error.
86h - not supported by BIOS configuration.

3.6.23 Set Console I/O Redirection (A1h)

The Set Console I/O Redirection BIOS function is called by application software to specify the device that will be used by the BIOS to redirect console input (INT 16h) and console output (INT 10h). This feature is only available in BIOS adaptations that provide for console redirection.

Console I/O can be redirected to the standard keyboard and screen with the device value, 0. Other values, such as 1, 2, 3, and so on, specify a COM port number that specifies the serial port that will be used. For example, the value 2 specifies that output will be redirected over the serial line attached to COM2.

Input Parameters:

AH - A1h, indicating the Set Console I/O Redirection Function.
BX - specifies the new console device. The value 0 indicates the standard keyboard and screen, and nonzero values indicate the COM port number (starting with 1 for COM1) to be used as a console redirection device.

Output Parameters:

CY - clear if success, else set if failure.
AH - status code, as follows:
00h - no error.
86h - not supported by BIOS configuration.

3.6.24 Get Embedded BIOS Version (A3h)

The Get Embedded BIOS Version BIOS function is called by application software to return the major and minor version codes of the underlying implementation of Embedded BIOS. This allows application software to determine if it can use specific features supported by the BIOS.

Input Parameters:

AH - A3h, indicating the Get Embedded BIOS Version Function.

Output Parameters:

CY - clear if success, else set if failure.
AL - for successful returns, the minor BIOS version (i.e., 0 for version 4.1).

AH - for successful returns, the major BIOS version (i.e., 4 for version 4.1).
AH - if failure, status code, as follows:

86h - not supported by BIOS configuration.

3.6.25 Get RFD Drive Information (A400h)

The Get RFD Drive Information BIOS function is called by Embedded DOS-ROM 6.22 to determine which INT 13h drive unit number is assigned to the RFD, and to obtain information about the size and shape of the RFD's underlying Flash array.

Input Parameters:

AX - A400h, indicating the Get RFD Drive Information Function.

Output Parameters:

CY - clear if success, else set if failure.
AX - if success, magic signature: ebfdh.
BX - size of Flash array in kilobytes.
CX - size of Flash block in kilobytes.
DI:SI - 32-bit media address of Flash array.
DL - RFD BIOS unit number.

AH - if failure, status code, as follows:

86h - not supported by BIOS configuration.

3.6.26 RFD Broadcast (A401h)

The RFD Broadcast BIOS function is called by Embedded DOS-ROM 6.22 to notify the RFD that specific sectors have been deallocated to free space and are no longer used to store user data. This allows the RFD software in the core BIOS to reclaim the area for storage of other data.

The RFD normally does not become aware of sectors previously written that have since become marked in the DOS file system FAT tables as being deallocated. This is because MS-DOS does not provide for such a notification method. Normally, programs such as Undelete can actually recover this lost data because it is not actually erased when a file is deleted on other media, such as hard drives or floppies; however, these media types do not incur performance penalties when the media retains the data.

In the case of the Resident Flash Disk, freeing up deallocated space can significantly improve file system performance; by as much as a factor of 10.

Input Parameters:

AX - A401h, indicating the RFD Broadcast Function.
CX - number of sectors to deallocate starting with the specified sector.
DX - starting sector number to deallocate.

Output Parameters:

CY - clear if success, else set if failure.
AH - if failure, status code, as follows:

86h - not supported by BIOS configuration.

3.6.27 Return System Configuration (C0h)

The Return System Configuration BIOS function is called to return the address of the System Configuration Table (SCT), as defined in Chapter 3. This table reveals the BIOS's support for various hardware features.

Input Parameters:

AH - C0h, indicating the Return System Configuration Function.

Output Parameters:

CY - clear if success, else set if failure.
AH - status code, as follows:

00h - no error.
86h - SCT not supported by BIOS configuration.

ES:BX - if success, 16:16 address of SCT data structure.

3.6.28 Return Extended BIOS Data Area (C1h)

The Return Extended BIOS Data Area BIOS function is called to return the 16-bit segment address of the EMBEDDED BIOS Extended BIOS Data Area, located in the top 1KB of low memory. The format of this area is General Software-proprietary.

Input Parameters:

AH - C1h, indicating the Return Extended BIOS Data Area Function.

Output Parameters:

CY - clear if success, else set if failure.
AH - status code, as follows:

00h - no error.
86h - function not supported.

ES - if success, segment address of Extended BIOS Data Area.

3.6.29 PS/2 Mouse Request (C2h)

The PS/2 Mouse Request BIOS function is called to process an application or operating system request to read the status of, or control the operation of, the PS/2 mouse.

Input Parameters:

AH - C2h, indicating the PS/2 Mouse Request Function.
AL - subfunction, as follows:

00h - Enable/Disable mouse.
01h - Reset mouse.
02h - Set sample rate.
03h - Set resolution.
04h - Get mouse type.
05h - Initialize mouse interface.
06h - Get mouse status/set scaling factor.
07h - Register callout address.

BH - sub-subfunction code or parameter for subfunction.

Output Parameters:

CY - clear if success, else set if failure.
AH - status code, as follows:

00h - no error.
01h - invalid subfunction code (code in AL.)
02h - invalid input value (value in BH.)
03h - I/O communications error.
04h - resend status received from mouse.
05h - no callout address registered.
86h - function not supported (if **OPTION_SUPPORT_PS2MOUSE** disabled.)

3.6.30 Watchdog Timer Control (C3h)

The Watchdog Timer Control BIOS function is called to enable or disable the watchdog timer, if available. When the watchdog timer is enabled by this function, the value passed in the BX CPU register is used as a countdown value for the timer. When the timer reaches 0, it resets the system with a warm boot.

If the timer is running and the disable subfunction is called, then the timer stops without resetting the system.

If the timer is running and the enable subfunction is called, the then the timer restarts with the specified value without resetting the system.

If the timer is stopped and the disable function is called, no operation is performed.

If the timer is stopped and the enable function is called, then the timer restarts with the specified value without resetting the system.

Input Parameters:

AH - C3h, indicating the Watchdog Timer Control Function.
AL - 00h to disable timer, 01h to enable timer.
BX - if enabling, fail-safe timer value (units unspecified).

Output Parameters:

CY - clear if success, else set if failure.
AH - status code, as follows:

00h - no error.
86h - function not supported.

3.6.31 Checksum Region (C4h)

The Checksum Region BIOS function is called to perform a BIOS checksum over a specified address range in low memory.

Input Parameters:

AH - C4h, indicating the Checksum Region Function.
DS:SI - 16:16 real-mode pointer to region to be checksummed.
CX - size of region in words.

Output Parameters:

CY - clear if success, else set if failure.
DX:AX - if successful, 32-bit checksum.

AH - if failure, status code, as follows:

86h - function not supported.

3.6.32 Debugger Breakpoint (D0h)

The Debugger Breakpoint BIOS function is called to perform a breakpoint into the debugger on systems where an operating system is loaded, and that operating system has revectored INT 3 to its own private dummy routine, otherwise making the BIOS debugger inaccessible.

Input Parameters:

AH - D0h, indicating the Debugger Breakpoint Function.

Output Parameters:

CY - clear if success, else set if failure.
AH - status code (if error), as follows:

86h - function not supported.

3.6.33 Flash Programming (E0h)

The Flash Programming BIOS function is called to perform any of a number of operations on the Flash array supported by the underlying BIOS. This function is General Software-proprietary.

There are four subfunctions, all of which require that the (DI:SI) register pair contain the 32-bit media address of Flash memory being manipulated. If the operation is lock or unlock, then (DI:SI) can point to any byte within the block to be locked or unlocked.
If the operation is a read or write, then this register pair points to the first byte in a contiguous area of Flash to be read or written.

For read and write operations, the (ES:BX) register pair points to a user buffer where the data read from Flash will be stored for read operations, or that will contain data to be written to Flash for write operations. The (CX) register is used to specify the number of bytes to transfer.

This function requires that Flash programming support (**OPTION_SUPPORT_MCL**) be enabled by the OEM adaptation. If this support is not enabled, this function will return CY set and AH=86h.

Input Parameters:

AH - E0h, indicating the Flash Programming Function.
AL - Subfunction, as follows:

00h - Lock block function.
01h - Erase block function.
02h - Read block function.
03h - Write block function.

DI:SI - 32-bit media address of Flash memory area.
CX - bytes to read or write (unused for lock or erase).

ES:BX - 16:16 real-mode address of a user buffer where information is transferred *from* on a write operation, or where it is transferred *to* on a read operation.

Output Parameters:

CY - clear if success, else set if failure.
AH - status code, as follows:

00h - no error.
86h - function not supported.

3.7 INT 16h, Keyboard Services

This section explains the keyboard BIOS application program interface (API). The keyboard BIOS is called through software interrupt 16H. Services are provided to read keystrokes from the keyboard typeahead buffer, peek at the next keystroke in the typeahead buffer, and get the status of the shift keys on the keyboard.

Additional services are provided by the INT 16h service module to set the CPU speed and manipulate the system's cache. These services are historically bound to the INT 16h service because they were first managed by the 8042 keyboard controller.

3.7.1 Read Keyboard Input (00h)

The Read Keyboard Input keyboard BIOS function is called to read a keystroke from the keyboard device, waiting until a keystroke arrives if one is not present. The scan code of the keystroke is returned in (AH), and the ASCII code is returned in (AL). An exception exists for function keys and ALT keys; in this case, the ASCII code returned is zero, and the scan code in (AH) is used to determine which function or ALT key was read from the keyboard.

Input Parameters:

AH - 00h, indicating the Read Keyboard Input Function.

Output Parameters:

AH - Scan code of the returned keystroke.
AL - ASCII code for the returned keystroke.

3.7.2 Return Keyboard Status (01h)

The Return Keyboard Status keyboard BIOS function is called to peek at the status of the typeahead buffer, to determine if a keystroke is waiting to be read. If not, the zero flag (ZF) is cleared, so that a JZ instruction after the INT 16H instruction would not be taken. If a keystroke is waiting to be read, then ZF is set, so that a JZ instruction after the INT 16H instruction would be taken. In the latter case, the scan code and character code are returned in (AH) and (AL), respectively. If a character is found, this function returns a copy of it but does not remove it from the typeahead buffer. The application can call function 00H to remove the character from the typeahead buffer properly.

Input Parameters:

AH - 01h, indicating the Return Keyboard Status Function.

Output Parameters:

ZF - Clear if character ready, else set if not.
AH - Scan code of the returned keystroke.
AL - ASCII code for the returned keystroke.

3.7.3 Return Shift Flag Status (02h)

The Return Shift Flag Status keyboard BIOS function is called to return the status of the shift keys, including the INS key, CAPS LOCK key, NUM LOCK key, SCROLL LOCK key, ALT key, CTRL key, and LEFT and RIGHT SHIFT keys.

Input Parameters:

AH - 02h, indicating the Return Shift Flag Status.

Output Parameters:

AL - Current shift status, in the form of a bit mask, one bit per shift key.

10000000b - INS key is active.
01000000b - CAPS LOCK key is active.
00100000b - NUM LOCK key is active.
00010000b - SCROLL LOCK key is active.
00001000b - ALT key is pressed down.
00000100b - CTRL key is pressed down.
00000010b - LEFT SHIFT key is pressed down.
00000001b - RIGHT SHIFT key is pressed down.

3.7.4 Set Typematic Rate (03h)

The Set Typematic Rate keyboard BIOS function is called to program the typematic delay and rate associated with holding down a key on the keyboard. This keyboard service is not redirectable over serial links.

Input Parameters:

AH - 03h, indicating the Set Typematic Rate Function.
AL - 05h
BH - typematic delay before repeat starts, as follows:

00h - 250 milliseconds.
01h - 500 milliseconds.
02h - 750 milliseconds.
03h - 1,000 milliseconds (1 second).

BL - typematic rate in characters per second, as follows:

00h - 30.0 CPS.	01h - 26.7 CPS.
02h - 24.0 CPS.	03h - 21.8 CPS.
04h - 20.0 CPS.	05h - 18.5 CPS.
06h - 17.1 CPS.	07h - 16.0 CPS.
08h - 23.1 CPS.	09h - 13.3 CPS.
0ah - 12.0 CPS.	0bh - 10.9 CPS.
0ch - 10.0 CPS.	0dh - 9.2 CPS.
0eh - 8.6 CPS.	0fh - 8.0 CPS.
10h - 7.5 CPS.	11h - 6.7 CPS.
12h - 6.0 CPS.	13h - 5.5 CPS.
14h - 5.0 CPS.	15h - 4.6 CPS.
16h - 4.3 CPS.	17h - 4.0 CPS.
18h - 3.7 CPS.	19h - 3.3 CPS.
1ah - 3.1 CPS.	1bh - 2.7 CPS.

1ch - 2.5 CPS.	1dh - 2.3 CPS.
1eh - 2.1 CPS.	1fh - 2.0 CPS.

Output Parameters:

none.

3.7.5 Push Data to Keyboard (05h)

The Push Data to Keyboard keyboard BIOS function is called to push data (a character and a scan code) into the keyboard typeahead buffer.

Input Parameters:

AH - 05h, indicating the Push Data to Keyboard Function.
CH - scan code to be pushed.
CL - character to be pushed.

Output Parameters:

CY - set if failure, else clear if success.
AL - error code, as follows:

00h - no error.
01h - keyboard buffer full.

3.7.6 Enhanced Read Keyboard (10h)

The Enhanced Read Keyboard keyboard BIOS function is called to read a character and scan code from the keyboard buffer when the BIOS supports an enhanced (101-key) keyboard. There is no advantage to calling this routine over the standard read function; it is provided for compatibility with some versions of DOS that call it.

Input Parameters:

AH - 10h, indicating the Enhanced Read Keyboard Function.

Output Parameters:

AH - 00h scan code or character ID if special character.
AL - ASCII code.

3.7.7 Enhanced Read Keyboard Status (11h)

The Enhanced Read Keyboard Status keyboard BIOS function is called to determine if an enhanced keyboard has a character waiting in its buffer. There is no advantage to calling this routine over the standard read function; it is provided for compatibility with some versions of DOS that call it. This routine does not remove the data from the keyboard buffer; it is a "peek" operation.

Input Parameters:

AH - 11h, indicating the Enhanced Read Keyboard Status Function.

Output Parameters:

ZF - set if no character, else clear if character waiting.

Then, if a character is waiting:

AH - 00h scan code or character ID if special character.
AL - ASCII code.

3.7.8 Enhanced Read Keyboard Flags (12h)

The Enhanced Read Keyboard Flags keyboard BIOS function is called to return the state of the enhanced shift flags maintained by 101-key keyboards. There is limited advantage to calling this routine over the standard read function; it is provided for compatibility with some versions of DOS that call it.

Input Parameters:

AH - 12h, indicating the Enhanced Read Keyboard Flags Function.

Output Parameters:

AX - 16-bit bitmask containing keyboard flags, as follows:

00000000.00000001 - right shift key pressed.
00000000.00000010 - left shift key pressed.
00000000.00000100 - ctrl key pressed.
00000000.00001000 - alt key pressed.
00000000.00010000 - scroll lock is on.
00000000.00100000 - num lock is on.
00000000.01000000 - caps lock is on.
00000000.10000000 - insert mode is on.
00000001.00000000 - left ctrl key is pressed.
00000010.00000000 - left alt key is pressed.
00000100.00000000 - right ctrl key is pressed.
00001000.00000000 - right alt key is pressed.
00010000.00000000 - scroll lock key is pressed.
00100000.00000000 - num lock key is pressed.
01000000.00000000 - caps lock key is pressed.
10000000.00000000 - sysreq key is pressed.

3.7.9 Set CPU Speed (F0h)

The Set CPU Speed BIOS function is called to change the CPU's clocking to either the low or high states.

Not all BIOS adaptations can switch speeds; this is largely a function of the BPM, CPM, and CSPM implementations.

Input Parameters:

AH - F0h, indicating the Set CPU Speed Function.
AL - speed to set, as follows:

00h - slow.
01h - medium.
02h - fast.

Output Parameters:

none.

3.7.10 Get CPU Speed (F1h)

The Get CPU Speed BIOS function is called to read the CPU's speed as set with the Set CPU Speed function.

Not all BIOS adaptations can switch speeds; this is largely a function of the BPM, CPM, and CSPM implementations.

Input Parameters:

AH - F1h, indicating the Get CPU Speed Function.

Output Parameters:

AL - current CPU speed, as follows:

00h - slow.
01h - medium.
02h - fast.

3.7.11 Read Cache Status (F400h)

The Read Cache Status BIOS function is called to request the status of the external cache as supported by the BIOS.

Not all BIOS adaptations can manipulate the cache; this is largely a function of the BPM, CPM, and CSPM implementations.

Input Parameters:

AH - F4h, indicating the Cache Control Function.
AL - 00h, indicating the Read Cache Status Subfunction.

Output Parameters:

AH - cache status, as follows:

not modified - no cache status is available.
E2h - successful, information returned.

AL - cache controller status, as follows:

00h - cache controller not present.
01h - cache memory enabled.
02h - cache memory disabled.

CX - cache memory size, as follows:

Bit 15 - 1 if information invalid, else 0 if valid.
Bits 14 through 0 - cache memory size in KB.

DH - cache write strategy, as follows:

Bit 7 - 1 if information invalid, else 0 if valid.
Bits 6 through 1 - set to 0's.
Bit 0 - 0 if write-through, else 1 if write-back.

DL - cache type, as follows:

Bit 7 - 1 if information invalid, else 0 if valid.
Bits 6 through 1 - set to 0's.
Bit 0 - 0 if direct-mapped else 1 if two-way set associative.

3.7.12 Enable Cache (F401h)

The Enable Cache BIOS function is called to enable the external cache controller as supported by the BIOS.

Not all BIOS adaptations can manipulate the cache; this is largely a function of the BPM, CPM, and CSPM implementations.

Input Parameters:

AH - F4h, indicating the Cache Control Function.
AL - 01h, indicating the Enable Cache Subfunction.

Output Parameters:

AH - cache status, as follows:

not modified - no cache status is available.
E2h - successful, operation performed.

3.7.13 Disable Cache (F402h)

The Disable Cache BIOS function is called to disable the external cache controller as supported by the BIOS.

Not all BIOS adaptations can manipulate the cache; this is largely a function of the BPM, CPM, and CSPM implementations.

Input Parameters:

AH - F4h, indicating the Cache Control Function.
AL - 02h, indicating the Disable Cache Subfunction.

Output Parameters:

AH - cache status, as follows:

not modified - no cache status is available.
E2h - successful, operation performed.

3.8 INT 17h, Parallel I/O Services

This section explains the parallel BIOS application program interface (API). The parallel BIOS is called through software interrupt 17H. Services are provided to write a character to the parallel port, initialize the printer attached to a parallel port, and read the status of a printer attached to a parallel port.

3.8.1 Write Character (00h)

The Write Character parallel BIOS function is called to write a character over the parallel port to a printer or other parallel device.

Input Parameters:

AH - 00h, indicating the Write Character Function.
AL - Character to print.
DX - Parallel port number (0=LPT1, 1=LPT2, 2=LPT3).

Output Parameters:

AH - Printer status, as follows:

10000000b - Printer not busy.
01000000b - Acknowledgement.
00100000b - Out of paper.
00010000b - Printer selected.
00001000b - I/O error occurred.
00000100b - Reserved.

00000010b - Reserved.
00000001b - Timeout error occurred.

3.8.2 Initialize Printer (01h)

The Initialize Printer parallel BIOS function is called to initialize an attached print device. It does this by pulsing the reset line on the parallel interface.

Input Parameters:

AH - 01h, indicating the Initialize Printer Function.
DX - Parallel port number (0=LPT1, 1=LPT2, 2=LPT3).

Output Parameters:

AH - Printer status, as follows:

10000000b - Printer not busy.
01000000b - Acknowledgement.
00100000b - Out of paper.
00010000b - Printer selected.
00001000b - I/O error occurred.
00000100b - Reserved.
00000010b - Reserved.
00000001b - Timeout error occurred.

3.8.3 Read Printer Status (02h)

The Read Printer Status parallel BIOS function is called to read the status lines attached driven by a parallel-mode printer attached to the parallel port.

Input Parameters:

AH - 02h, indicating the Read Printer Status Function.
DX - Parallel port number (0=LPT1, 1=LPT2, 2=LPT3).

Output Parameters:

AH - Printer status, as follows:

10000000b - Printer not busy.
01000000b - Acknowledgement.
00100000b - Out of paper.
00010000b - Printer selected.
00001000b - I/O error occurred.
00000100b - Reserved.
00000010b - Reserved.
00000001b - Timeout error occurred.

3.9 INT 1ah, Time Services

This section explains the date/time BIOS application program interface (API). The date/time BIOS is called through software interrupt 1aH. Services are provided to read the system time counter, write the system time counter, read the real time clock, write the read time clock, read the real time clock date, and write the real time clock date.

3.9.1 Read System Timer Count (00h)

The Read System Timer Count date/time BIOS function is called to return the 32-bit number of ticks since last midnight as stored in the BIOS data area.

Input Parameters:

AH - 00h, indicating the Read System Timer Count Function.

Output Parameters:

CY - set if failure, else clear if success.
AH - 00h.
AL - Timer overflow flag:

00h - Time has not overflowed the field.
01h - Time has overflowed the field.

CX:DX - 32-bit number of ticks elapsed since midnight.

3.9.2 Write System Timer Count (01h)

The Write System Timer Count date/time BIOS function is called to store the 32-bit number of ticks since last midnight into the BIOS data area.

Input Parameters:

AH - 01h, indicating the Write System Timer Count Function.
CX:DX - 32-bit number of ticks elapsed since midnight.

Output Parameters:

CY - set if failure, else clear if success.
AH - 00h.

3.9.3 Read Real Time Clock Time (02h)

The Read Real Time Clock Time date/time BIOS function is called to return the time information from the battery-backed real time clock.

Input Parameters:

AH - 02h, indicating the Read RTC Time Function.

Output Parameters:

AH - 00h.
CY - set if RTC update in progress, else clear if success.
CH - Hours in BCD.
CL - Minutes in BCD.
DH - Seconds in BCD.
DL - Daylight savings option:
    00h - No daylight savings supported.
    01h - Daylight savings supported.

3.9.4 Write Real Time Clock Time (03h)

The Write Real Time Clock Time date/time BIOS function is called to store the time information into the battery-backed real time clock.

Input Parameters:

AH - 03h, indicating the Write RTC Time Function.
CH - Hours in BCD.
CL - Minutes in BCD.
DH - Seconds in BCD.
DL - Daylight savings option:
    00h - No daylight savings supported.
    01h - Daylight savings supported.

Output Parameters:

AH - 00h.
AL - Value written to CMOS 0bh register.
CY - set if RTC update in progress, else clear if success.

3.9.5 Read Real Time Clock Date (04h)

The Read Real Time Clock Date date/time BIOS function is called to return the date information from the battery-backed real time clock.

Input Parameters:

AH - 04h, indicating the Read RTC Date Function.

Output Parameters:

AH - 00h.
CY - set if RTC update in progress, else clear if success.
CH - Century in BCD (19h or 20h).
CL - Year in BCD.

DH - Month in BCD.
DL - Day in BCD.

3.9.6 Write Real Time Clock Date (05h)

The Write Real Time Clock Date date/time BIOS function is called to store the date information into the battery-backed real time clock.

Input Parameters:

AH - 05h, indicating the Write RTC Date Function.
CH - Century in BCD (19h or 20h).
CL - Year in BCD.
DH - Month in BCD.
DL - Day in BCD.

Output Parameters:

AH - 00h.
AL - Value written to CMOS 0bh register.
CY - set if RTC update in progress, else clear if success.

3.9.7 PCI Services (B1h)

The PCI Services BIOS function is called to make a PCI function request. The PCI API is well-documented by the PCI Consortium and its operation is beyond the scope of this section.

Input Parameters:

AH - b1h, indicating the PCI Services function.
Others - as defined by PCI Specification.

Output Parameters:

All - as defined by PCI Specification