A new computer architecture that includes an information management executive for coupling memory based peripheral storage units to a computer operating system that typically uses I/O commands for addressing peripheral storage units is disclosed. The information management executive includes an I/O executive, a memory mapping executive and a memory mapper. According to the principles of this invention, the information management executive receives both I/O mapped commands and memory mapped commands from the operating system and either retrieves or stores the information specified in the operating system command in the memory of a memory based peripheral storage unit.
DESIGNATIONS OF “DE”

Until further notice, any designation of “DE” in any international application whose international filing date is prior to October 3, 1990, shall have effect in the territory of the Federal Republic of Germany with the exception of the territory of the former German Democratic Republic.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>AT</td>
<td>Spain</td>
<td>ES</td>
<td>Monaco</td>
<td>MC</td>
</tr>
<tr>
<td>Australia</td>
<td>AU</td>
<td>Finland</td>
<td>FI</td>
<td>Madagascar</td>
<td>MG</td>
</tr>
<tr>
<td>Barbados</td>
<td>BB</td>
<td>France</td>
<td>FR</td>
<td>Mali</td>
<td>ML</td>
</tr>
<tr>
<td>Belgium</td>
<td>BE</td>
<td>Gabon</td>
<td>GA</td>
<td>Mauritania</td>
<td>MR</td>
</tr>
<tr>
<td>Burkina Faso</td>
<td>BF</td>
<td>United Kingdom</td>
<td>GB</td>
<td>Malawi</td>
<td>MW</td>
</tr>
<tr>
<td>Bulgaria</td>
<td>BG</td>
<td>Greece</td>
<td>GR</td>
<td>Netherlands</td>
<td>NL</td>
</tr>
<tr>
<td>Benin</td>
<td>BJ</td>
<td>Hungary</td>
<td>HU</td>
<td>Norway</td>
<td>NO</td>
</tr>
<tr>
<td>Brazil</td>
<td>BR</td>
<td>Italy</td>
<td>IT</td>
<td>Poland</td>
<td>PL</td>
</tr>
<tr>
<td>Canada</td>
<td>CA</td>
<td>Japan</td>
<td>JP</td>
<td>Romania</td>
<td>RO</td>
</tr>
<tr>
<td>Central African Republic</td>
<td>CF</td>
<td>Democratic People's Republic of Korea</td>
<td>KP</td>
<td>Sudan</td>
<td>SD</td>
</tr>
<tr>
<td>Congo</td>
<td>CG</td>
<td>Republic of Korea</td>
<td>KR</td>
<td>Sweden</td>
<td>SE</td>
</tr>
<tr>
<td>Switzerland</td>
<td>CH</td>
<td>Liechtenstein</td>
<td>LJ</td>
<td>Senegal</td>
<td>SN</td>
</tr>
<tr>
<td>Cameroon</td>
<td>CM</td>
<td>Sri Lanka</td>
<td>LK</td>
<td>Soviet Union</td>
<td>SU</td>
</tr>
<tr>
<td>Germany</td>
<td>DE</td>
<td>Luxembourg</td>
<td>LU</td>
<td>Chad</td>
<td>TD</td>
</tr>
<tr>
<td>Denmark</td>
<td>DK</td>
<td></td>
<td></td>
<td>Togo</td>
<td>TG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>United States of America</td>
<td>US</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A METHOD AND APPARATUS FOR INFORMATION MANAGEMENT IN A COMPUTER SYSTEM

BACKGROUND OF THE INVENTION

5 Field of the Invention

This invention relates generally to computer systems and more specifically to a computer system having multiple memory cards configured as disk drives in which the memory cards function as either disks or expanded memory.

10 Prior Art

Many different computing systems are available today but most of these systems are built around fundamental components such as those illustrated in Figure 1. Typically the fundamental components of a computer 20, include a central processing unit 10 (CPU) which is connected through an input bus 11 to an input module 12 and through an output bus 13 to an output module 14. CPU 10 is also connected through data buses 15, 16 to a memory unit 17.

20 CPU 10 provides control and computing functions. Input and output modules 12, 14 are used to communicate between the computer user and CPU 10. Input module 12 supplies information to CPU 10. Typical input devices are a keyboard and a mouse. Output module 14 displays information from central processing unit 10. Typical output modules include a video display monitor, a printer and other visual display means such as plotters. Input unit 12 and output unit 14 are frequently referred to as input/output (I/O) units.

30 Memory unit 17 typically contains two general types of information, computer programs and data. A computer program is a sequence of instructions that are executed by the computer to perform a specific function. Data in
memory unit 17 are processed by CPU 10 in response to the instructions from a computer program which is executing in CPU 10.

Memory unit 17 typically includes mass memory 17A, sometimes called secondary memory, and main memory 17B. Main memory 17B is a relatively fast memory, i.e. a typical access time is in the range from 20 to approximately 400 nanoseconds. Access time is the time interval between when CPU 10 requests data from memory 17 and when memory 17 makes the requested data available to CPU 10.

Main memory 17B is usually used to store at least a portion of the program currently being executed by CPU 10 and data required by this program. Mass memory 17A, such as disks and tapes, is used to store programs, data, or portions of either programs and data which are not needed immediately by CPU 10 or which cannot be accommodated in main memory 17B because of size limitations of main memory 17B. Since programs and/or data are transferred in and out of mass memory 17A at the direction of CPU 10, mass memory units are typically included in the generic term "I/O units." Mass memory 17A, is significantly slower than main memory 17B. Access time for mass memory is typically on the order of tens of milliseconds.

A computer operating system is typically used to control the operation of CPU 10 (Fig. 1), main memory 17B and I/O modules 12, 14, 17A. In addition, the operating system provides an interface between computer user applications and the hardware. As used herein, hardware refers to the physical components of a computer system. The operating system of a computer typically includes a kernel which (i) creates user processes, as described below, (ii) schedules execution of user processes, (iii) provides user processes with system services, and (iv) services hardware interrupts and exceptions. The computer operating system is usually loaded into main memory 17B when the computer is booted. As used herein,
"booted" refers to the sequence of operations that are performed when either power is first applied to the computer or the computer is reset.

The definition of a "user process" requires an understanding of the sequence of steps by which user source code, i.e., a computer program, is transformed into a series of instructions for CPU 10. User source code for an application program, which is to be run on a computer using a particular operating system, is typically compiled and linked into a binary file which is known as an executable image. A "user process" is execution of the executable image by the kernel of the operating system.

In a computer operating under a system such as Microsoft Corporation's MS-DOS operating system (MS-DOS), the interface between the operating system and hardware such as the keyboard, the display screen, and the disks are included in a basic input/output system (BIOS), which is usually contained in a read-only memory (ROM) within computer 20. (MS-DOS is a registered trademark of Microsoft Corporation.) The BIOS is designed to drive specific hardware based upon instructions, sometimes called commands, from the operating system.

More specifically, the BIOS interfaces with the operating system through a series of commands called interrupts. The interrupts are usually initiated by software executing in CPU 10. In response to an interrupt, CPU 10 obtains an address of the interrupt service routine from an interrupt vector table stored in main memory 17B. Typically, the address of an interrupt service routine points to areas of main memory 17B which contains ROM BIOS and the operating system where the interrupt service routine is stored. Thus, main memory 17B typically contains user processes, an operating system, and information to interface the operating system with the hardware coupled to CPU 10.

As an example, main memory 17B in an IBM microcomputer or an IBM compatible microcomputer,
sometimes referred to as the address space, typically consists of one Mbyte and the operating system is loaded into the lower region of memory 17B. Figures 2A and 2B illustrate the standard memory map for main memory 17B under the MS-DOS operating system. The 1 Mbyte is subdivided into sixteen 64 Kbyte segments 17B-1 to 17B-16.

As illustrated in Figures 2A and 2B, the lower memory segments 17B-1 to 17B-10 are user area 25. Programs, which are executed by computer 20, must fit within a maximum of 640 Kbyte user area 25 or portions of the program must be swapped between user area 25 and mass memory 17A.

Figure 2B illustrates in more detail a typical configuration of user memory area 25. Specifically, the interrupt vector table is loaded in the lowest part of memory area 25, followed by the BIOS data area, followed by the DOS operating system kernel including DOS buffers and DOS device drivers and the file COMMAND.COM. Each of these regions are known to those skilled in the art and are explained more completely in P. Norton and R. Wilton, The New Peter Norton Programmer's Guide to The IBM PC & PS/2, Microsoft Press, (1988). The important aspect is that the BIOS data area occupies a specific region of lower memory as does the operating system. The BIOS data area contains specific bits of information which can be interrogated to ascertain the status of computer system 20.

Memory segments 17B-11 to 17B-16 (Fig. 2A), with addresses extending from A0000 to FFFFF, are used by computer system 20, and are usually referred to as system area 26. Specifically, memory segments with addresses ranging from A0000 through BFFFF are reserved for graphic video displays and the segment with addresses ranging from F0000 to FFFFF is for system ROM BIOS, which is described more completely below. The 192 Kbyte region extending from addresses C0000 to EFFFF does not have a well defined standard that associates this region with any particular
use. However, in the IBM AT computer architecture, the system ROM typically extends from E0000 to address FFFFF.

In addition to main memory 17B, computer 20 may also include expanded memory 27, as illustrated in Fig. 3. Expanded memory 27 is outside the address space of CPU 10. A device driver in the operating system typically maps segments of memory 27, sometimes called pages, into a segment of memory 17B.

The LIM EMS 4.0 is one standard for use of expanded memory. A copy of the LIM EMS 4.0 expanded memory specification is available from Intel Corp. of Santa Clara, California. Under the LIM EMS specification, the individual segments, i.e., pages, in memory 27 are 16 Kbyte in size. The LIM EMS specification provides a standard interrupt with functions which specify operations to be performed between expanded memory 27 and main memory 17B. The LIM EMS device driver copies pages from expanded memory 27 into segments 17B-13 to 17B-15 in system area 26 of main memory 17B and subsequently into transient user area 25A. Segments 17B-13 to 17B-15 are referred as the page frame for expanded memory 27.

In some applications, pages in the page frame are used for data storage for a program executing from user area 25 and the pages are swapped in and out of expanded memory by the executing program. If an executable image of a user program is stored in expanded memory 27, the executable image is typically moved from the page frame to the user area 25 for execution.

An expanded memory board is usually plugged into an expansion slot in computer 20 and is effectively a permanent attachment to computer 20. The board is generally not designed to be inserted and removed from computer 20 in the same manner as say a floppy disk, or a tape in a tape drive. Accordingly, floppy disks and hard disks are typically used either to provide data and executable images of user applications to computer 20 or to store data for computer 20.
Thus, most operating systems and in particular the MS-DOS operating system, are configured for I/O mapped mass memory 17A, i.e., structures having a geometrical configuration such as a disk. A typical disk drive consists of one or more platters. The surfaces of each platter are coated with a magnetic material and each coated surface has a read and write head. Each surface of a platter is organized into a series of concentric tracks and each track is further subdivided into a group of sectors.

The disk drive is designed to position the read/write heads over a cylinder where a cylinder is an imaginary structure that passes through the same track on each platter. Thus, when the heads are moved to the second cylinder, the head for each surface is positioned over the second track on that surface. A particular head is selected for reading or writing a particular sector on the selected track. Thus, the interrupt for disks, i.e., I/O mapped structures, provides information about the disk drive, the head, track, and sector to be read or written. As used herein, an "I/O mapped structure" refers to a structure such as a disk that is normally addressed by specifying a head, track and sector, while a "memory mapped structure" refers to a memory structure that is normally addressed with a memory address such as those used for random access memory.

Unfortunately, both floppy disk drives and hard disk drives consume large amounts of power, on the order of 10 to 15 watts in some cases. Such devices are not easily incorporated in a portable computer which is driven solely by batteries. Moreover, the physical size of disk drives, typically either 3 1/2 inches or 5 1/4 inches, consumes considerable space within a portable computer. The heat generation of the drives coupled with the large size introduces both size and power constraints for portable computers.

Recognizing the limitations of floppy drives and hard
disk drives with respect to portable computers, the industry is developing smaller drives which have reduced power requirements. However, even if smaller low power disk drives become widely available, the performance of the computers will be limited by the disk capacity and the disk access time. Disk drives have slow access times in comparison to the access time for memory and therefore degrade performance of the computer when the disk is frequently accessed by a program.

Virtual RAM disks are sometimes formed using, for example, the portion of system memory 26 from address C0000 to address F0000. Alternatively, expanded or extended memory has also been configured as a virtual RAM disk. A virtual RAM disk is volatile and subject to losing its state should the power to the computer be interrupted. Nonvolatile RAM disks have been developed for some disk controllers which in turn interface with a driver in the operating system of the computer. Accordingly, such devices are not suitable for directly incorporating into the operating system of a computer. Moreover, since the nonvolatile RAM disks are addressed through the disk controller, the nonvolatile RAM cannot be utilized directly, for example, as expanded memory.

SUMMARY OF THE INVENTION

According to the principles of this invention, a novel computer architecture is provided wherein peripheral storage devices are utilized in response to both I/O mapped commands and memory mapped commands from the computer operating system. That is, the same peripheral storage devices are used independently of whether the operating system issues an I/O mapped command that specifies for example, head, cylinder and track location, or a memory mapped command that specifies a memory address similar to that typically used in addressing either random access memory or read-only memory.

The novel computer architecture of my invention
includes an information management executive that functions as an interface between a computer operating system and peripheral storage devices attached to a computer system. The information management executive, in one embodiment, includes an I/O executive, a memory mapping executive, and a memory mapper. The peripheral storage devices include a random access memory and a read-only memory. Thus, a computer system including the information management executive uses nonvolatile random access memory cards, read-only memory cards or combinations of read-only memory cards and nonvolatile random access memory cards as peripheral storage devices.

All commands from the operating system for peripheral storage devices are processed by the information management executive and supplied to either the I/O executive or the memory mapping executive. The I/O executive receives the I/O mapped commands issued by the computer operating system and converts these commands to memory mapped command. After converting the I/O mapped command to a memory mapped command, the I/O executive performs the memory mapped command using one of the peripheral storage devices.

For example, if the I/O mapped command from the operating system is to store a block of information on a peripheral storage device starting at a specified head, track, and sector location, the I/O executive converts the head, track and sector location to a memory address. The I/O executive also determines whether the requested peripheral storage device is coupled to the computer and whether the storage device can support the command issued by the operating system. If the I/O executive determines that these conditions are true, the I/O executive performs the memory mapped command using the appropriate peripheral storage device.

The I/O executive of this invention supports all of the prior art disk services. Therefore, prior art user applications and procedures are used without modification
in a computer system having the architecture of this invention. The novel information management executive opens a completely new capability in portable computers. The RAM cards and ROM cards, which function as peripheral storage devices, are small in size, durable, and easily transportable. The memory cards eliminate the need for power hungry disk drives. The access time for the RAM and or ROM cards is several orders of magnitude faster than the access time for a floppy disk or a hard disk. This feature significantly enhances the performance of the computer for user applications which frequently access mass memory.

Not only does the information management executive of this invention permit addressing memory based structures with I/O mapped commands from the operating system, but also the memory mapping executive included in the information management executive provides a means for using the memory based peripheral devices as expanded memory. As used herein, "memory based devices" includes only devices having either a random access memory or a read-only memory and does not include disk drives.

In response to an operating system memory mapped command, the memory mapping executive locates the requested information and either maps the information from main memory to the specified peripheral storage device or maps the information from the specified peripheral storage device to main memory. Hence, the information management executive with the I/O executive and the memory mapping executive permits the computer operating system to use a secondary memory device as either an I/O mapped device or a memory mapped device. Prior art computer systems generally utilize secondary memory only in an I/O mapped capacity.

Moreover, the computing operating system can address the secondary memory devices, according to the principles of the invention, using any sequence of I/O mapped commands and memory mapped commands. Specifically, memory
in a peripheral storage device is not reserved by the operating system for either I/O mapped commands or memory mapped commands, but rather the information management executive uses the memory for both types of commands without regard to previous or subsequent commands so long as the instant command is valid.

The memory based peripheral storage devices of this invention are formatted with a logical structure that facilitates retrieval of the information stored in the devices. Specifically, in one embodiment the logical structure includes a boot sector, a file access table, a root directory and a data region which corresponds to the tracks and sectors on a normal disk. This logical structure is organized so that there is a direct correspondence between the size of the storage area on the prior art disk and the size of each area in memory. In another embodiment, read-only memory structures have a logical structure which identifies the number of executable images which are stored in the read-only memory and the location of each executable image in the memory. In yet another embodiment, the read-only memories includes a logical structure which has both the boot sector, file access table, root directory structure and the executable image structure.

The computer architecture of this invention enhances computer performance and lowers the power requirements of the computer system in comparison to prior art computer systems. Thus, the operation of portable computers is no longer limited by the power requirements of the disk drives.

DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a prior art computer system.
Figures 2A and 2B illustrate a typical main memory map in a prior art microcomputer.
Figure 3 illustrates prior art expanded memory.
Figure 4 illustrates a computer system according to
the principles of this invention.

Figure 5 illustrates a main memory map for a computer system according to the principles of this invention.

Figure 6 illustrates the logical structure of a non-volatile random access memory card and a read-only memory card according to the principles of this invention.

Figures 7A and 7B illustrate other logical structures of a read-only memory card according to the principles of this invention.

Figure 8 is a flow diagram that illustrates disk operations using the I/O executive of this invention.

Figure 9 illustrates a disk operation performed by the I/O executive of this invention.

Figure 10 is a flow diagram that illustrates expanded memory operations using the memory mapping executive of this invention.

Figure 11 illustrates a memory swap operation performed by the memory mapping executive of this invention.

Figure 12 is a flow diagram for selected functions in the I/O executive of this invention.

Figure 13 is a flow diagram of the choose segment operation of Figure 12.

Figure 14 is a flow diagram for interrupt 66h in the memory mapper of this invention.

Figure 15 is a flow diagram of function 5 of the memory mapping executive of this invention.

Figure 16 is a flow diagram of function 4 of the memory mapping executive of this invention.

Figure 17 is a flow diagram of function 6 of the memory mapping executive of this invention.

DETAILED DESCRIPTION

According to the principles of this invention, a new information management executive is added to a computer having (i) a main memory which is configured as a user area and a system area and (ii) an operating system for
use with I/O mapped secondary memory. This information management executive includes, as described more completely below, an I/O executive that translates I/O mapped commands issued by the operating system for secondary memory to memory mapped commands so that either RAM or ROM in a peripheral storage device can be used as secondary memory. Thus, a user can access secondary memory using the same programs and commands as in prior art computer systems even though the secondary memory consists only of ROM or non-volatile RAM and not disks.

The information management executive of this invention also includes means for retrieving an executable image of a user application from a peripheral storage device consisting of either nonvolatile random access memory or a read-only memory and loading that executable image into the system area of main memory. The information management executive’s ability to load an executable image of a user application from a read-only memory into the addressable memory space of a computer system permits use of ROM cards to replace the prior art write protected floppy disk. In fact, the information management executive of this invention transparently provides all the disk services of prior art computer systems even though memory is used in place of the disk.

In contrast to prior art computer systems, which mapped an executable image from expanded memory into system area and subsequently transferred the image from the system area into the user area, according to the principles of this invention, the executable image may be executed from either the system area or from the user area. This provides many advantages over the prior art configurations. For example, if a spreadsheet program is executed in the system area, the user area is available as data area. This configuration permits a much larger spreadsheet than in the prior art applications.

Not only does the information management executive of this invention permit addressing memory based structures
with I/O mapped commands, but also the executive includes
means, i.e., a memory mapping executive described below,
for using secondary memory as expanded memory. (As used
herein, "memory based structures" or similar phrases refer
to structures having, for example, a random access memory
or a read-only memory and not structures having a disk.)
Thus, expanded memory is no longer limited to memory on a
board mounted in an expansion slot as in the prior art
computer systems. This information management executive
with the I/O executive and the memory mapping executive
permits using a secondary memory device as either an I/O
mapped device or a memory mapped device. Prior art
computer systems generally utilized secondary memory in an
I/O mapped capacity only.

The novel information management executive of this
invention opens a completely new capability in portable
computers. Since, according to the principles of this
invention, either a nonvolatile RAM or a ROM replaces the
floppy disks and the hard disk in a computer system, the
power requirements of the computer system are signific-
antly reduced and the performance of the computer system
is significantly improved. Small credit card sized
nonvolatile RAM memories are available. In a computer
system containing the novel information management
executive of this invention, these memories replace the
disk drives. In one embodiment of this invention, the
unique information management executive and the credit
card sized nonvolatile RAM are utilized in conjunction
with a low power portable computer, which is described in

pending, commonly assigned and commonly filed
application of John P. Fairbanks, et al., entitled
"Portable Low Power Computer", U.S. Serial No. 07/375,721,
which is incorporated herein by reference.

There are several other distinctive advantages of a
computer system having the novel information management
executive of this invention. First, the RAM cards and the
ROM cards are small in size, durable, and easily
transportable. The memory cards eliminate the need for power hungry disk drives. Second, the access time for the RAM and/or ROM cards is several orders of magnitude faster than the access time for either a floppy disk or a hard disk. The improved access time for mass memory significantly enhances the performance of the computer for applications which frequently access mass memory. Unlike the prior art expanded memory which mapped selected pages from expanded memory on an expanded memory board to the user area, according to the principles of this invention as described below, information from any secondary memory RAM device, any secondary memory ROM device, or an expanded memory board may be mapped into either the user area or the system area of main memory.

In one embodiment of this invention, as illustrated in Figure 4, computer system 100 includes CPU 107 which is coupled to (i) main memory 101, (ii) a read-only memory (ROM) 102, (iii) a random access memory (RAM) 103, (iv) a first memory card 104 and (v) a second memory card 105 by bus 106. Bus 106 represents the conventional data and address buses used in a microcomputer to interface a CPU with a memory. Computer system 100 also includes a keyboard and a display device, and is couplable to a printer, but these peripheral I/O units as well as other components of computer 100 are not illustrated in Figure 3 for clarity. The other peripheral I/O units and components required for computer system 100 are known to those skilled in the art.

ROM 102, RAM 103, and memory cards 104, 105 comprise the secondary memory of computer 100 and are sometimes referred to as "peripheral storage devices". In contrast to prior art computer 20, (Fig. 1) which usually used a plurality of disk drives as secondary memory, the secondary memory of computer 100, i.e., memories 102-105, has an access time on the same order of magnitude as the access time of main memory 101. Therefore, computer 100 has significantly enhanced I/O operation with respect to
prior art microcomputers. Further, the power consumption of the secondary memory is significantly less than the power consumption of the prior art disk drives so that the secondary memory of computer 100 is suitable for use in a battery powered computer.

In one embodiment, the information management executive 111 of this invention includes a memory mapping executive 108, an I/O executive 109, and a memory mapper 110. Computer system 100 also includes a ROM BIOS for the peripheral devices coupled to CPU 100 such as a display screen, a keyboard, and a printer. In one embodiment, the ROM BIOS is functionally equivalent to the ROM BIOS typically used in the IBM XT computer architecture and the operating system is the MS-DOS operating system. One embodiment of the ROM BIOS is presented in copending, commonly assigned, and commonly filed application of John P. Fairbanks, et al., entitled "Portable Low Power Computer", U.S. Serial No.07/375,721, which is incorporated herein by reference.

CPU 107, in this embodiment, is the Harris 80C88 microprocessor available from Harris Corp. of Melbourne, Florida. When computer system 100 is booted, novel information management executive 111 of this invention is loaded from a ROM in computer system 100 into the ROM BIOS region of main memory 101 along with the ROM BIOS. In another embodiment, memory management executive 108 is loaded as a terminate and stay resident program and the remaining portions of information management executive 111 are loaded from a ROM into the ROM BIOS region of main memory 101. The loading of the ROM BIOS and the ROM based information management executive 111 is functionally equivalent to the normal loading operation of an IBM XT computer system and is known to those skilled in the art. However, in computer system 100, the operating system has been modified so that computer 100 can be booted from memory cards 104, 105 or ROM 102. As explained more completely below, memory cards 104, 105 correspond to
floppy disk drives A and B in a microcomputer and ROM 102 corresponds to a write protected hard disk drive C.

While the use of memories 102-105 for secondary memory provides many advantages, most operating systems and in particular the MS-DOS operating system, are not designed to interface with either read-only memory or random access memory as mass memory but rather are configured to interact with I/O mapped structures, as previously described. Nevertheless, computer 100 is compatible with the MS-DOS operating system and software that executes under MS-DOS.

Specifically, I/O executive 109, as described more completely below, accepts I/O mapped commands from the operating system, which is loaded in main memory 101, and either reads the appropriate data from one of secondary memory devices 102-105 or writes the appropriate data to one of secondary memory devices 102-105. Similarly, memory mapping executive 108 and memory mapper 110 process memory mapping commands from user applications and perform the requested operation using the appropriate secondary memory device 102-105, as described below.

In some prior art systems, if expanded or extended memory was configured as a virtual disk, the memory was reserved for disk operations and therefore was not available for memory mapping operations. To use the reserved memory for other than disk operations, the user redefined the operating system so that the memory region was no longer reserved. In contrast, according to the principles of this invention for any memory based storage unit, a first operating system command for a memory mapping can be immediately followed by either another memory mapping command, a disk read command or a disk write command, for example, without any modification to the operating system by the user. Thus, unlike the prior art, the computer architecture of this invention does not require reserving a first region of memory in the peripheral storage devices for I/O mapped commands and a
second region of memory, which is different from the first region, for memory mapped commands.

Memory mapping executive 108 also removes the prior art limitation that restricted user applications to user program area 125 of main memory 101. Specifically, the user may access not only the 640 Kbyte user area 125 but also at least 192 Kbyte of system area 126, as described more completely below. Thus, the novel computer architecture provided by memory mapping executive 108 and memory mapper 110 between the operating system and the hardware of computer 100 not only improves the range of devices which can be accessed for memory mapping by the operating system, but also enhances the operational capability of computer 100 by making at least about 832 Kbytes of addressable main memory 101 available for user applications.

A main memory 101 map for one embodiment of computer system 100 is given in Fig. 5. The address for the various regions in main memory are given on the left-hand side of the memory map. User area 125 extends from address 0000:0000 to address 9000:FFFF while the system area 126 extends from address A000:0000 to address F000:FFFF. Several functional regions are defined within system area 126. Specifically, the region between addresses A000:0000 and A000:FFFF is for EGA graphics; between addresses B000:0000 and B000:OFFF is for monochrome graphics; between addresses B800:0000 and BB00:00FF is for CGA graphics; between address BC00:0000 and B000:FFFF is for a LCD pixel display; between F000:C000 and FFFF:0000 is for ROM BIOS; and address FFFF:0000 is the power on reset vector.

The segments between addresses C000:0000 and C000:FFFF, D000:0000 and D000:FFFF, E000:0000 and E000:FFFF, and F000:0000 and F000:FFFF are available for memory mapping as described below.

The operation of computer 100 is substantially similar to a standard microcomputer having floppy disk
drives A and B and a hard disk partitioned as drives C and D. In computer 100, memory cards 104, 105 function as the floppy disk drives. In one embodiment, memories cards 104 and 105 may be either read-only memory cards or non-volatile random access memory cards. The read-only memory cards function as a write protected floppy disk drive while the nonvolatile random access memory cards function in the same manner as a floppy disk drive.

Specifically, the user may write data to memory cards 104, 105, load data from memory cards 104, 105 into computer 100, or perhaps use the memory capacity of memory cards 104, 105 as storage space in execution of a user application. Moreover, unlike the expanded memory cards of the prior art computer which plugged into an expansion slot and were intended to be permanently affixed to the microcomputer, memory cards 104, 105 are designed to be removed and easily interchanged in the same manner that floppy disks are normally used. In this embodiment, read-only memory 102, which is equivalent to a C drive in the prior art machine, and nonvolatile random access memory 103, which is equivalent to a D drive in the prior art machine, are mounted within computer 100.

Memory cards 104, 105 are nonvolatile RAM cards, which have address and data buses compatible with the address and data buses of computer 100. In one embodiment, 512 Kbyte static RAM cards 104, 105 suitable for use in this invention are sold by Fujitsu Limited of Japan under the part number 9090-30 8904 P00 ES. This memory card is compatible with either a computer architecture having an 8-bit data bus and a 20-bit address bus or a computer architecture having a 16-bit data bus and a 20 or 24-bit address bus. Since, in one embodiment, computer system 100 has an 8-bit data bus and a 20-bit address bus, a signal is applied to the memory card from computer 100 according to the manufacturer’s specification to alternately select 8-bits on the 16-bit data bus, i.e., the information on bit lines 0-7 are used as a first 8-bit
word and then the information on bit lines 8-15 are used as a second 8-bit word. Configuring a 16-bit data bus to an 8-bit data bus is known to those skilled in the art because interchangeable boards are made for the IBM XT computer architecture which has an 8-bit data bus and for the IBM AT computer architecture which has a 16-bit data bus.

Read-only memory 102 contains the operating system for computer 100 as well as executable images of application programs. For example, a spreadsheet executable image or a word processing executable image could be loaded in ROM 102. In one embodiment, ROM 102 is a mask programmed ROM that is supplied by Poqet Computer Corp. of Sunnyvale, California under part number 15-0218-00. The operating system and any other executable images desired on ROM 102 are loaded in ROM 102 using techniques known to those skilled in the art. However, as described more completely below, in entering the images of these programs and/or applications, the ROM is configured with a specific logical structure.

In one embodiment, random access memory 103, which functions as the D drive in computer 100, is sixteen non-volatile static RAMs with each RAM having a capacity of 256 Kbit. A nonvolatile RAM suitable for use with this invention is supplied by Fujitsu Limited of Japan under part number MB84256-10LL-PF.

In a conventional microcomputer, such as computer 20 (Fig. 1) operating under the MS-DOS operating system, the operating system is contained on a hard disk drive, typically the C drive, along with files CONFIG.SYS and COMMAND.COM. The files CONFIG.SYS and COMMAND.COM are known to those skilled in the art. (See for example, Microsoft MS-DOS User's Guide and User's Reference which is supplied by Microsoft Corp. with the purchase of the MS-DOS operating system.) In computer system 100, the CONFIG.SYS file, as in the prior art systems, is supplied by the user and is used to configure the operating system.
However, in computer 100 the C drive, which contains the operating system is ROM 102. The CONFIG.SYS file is not included on ROM 102 because ROM 102 does not provide a capability for user modification of the CONFIG.SYS file. Accordingly, this file is maintained on the D drive, i.e., the nonvolatile random access memory 103. In this embodiment, the MS-DOS operating system is modified, so that the CONFIG.SYS file may be on the D drive while the operating system is on the C drive. Specifically, the IO.SYS file of MS-DOS is modified so that instead of looking to the C drive for the CONFIG.SYS file, the IO.SYS file looks to the D drive for this file when the computer is booted. With this change to the operating system, computer 100 is booted and reset in a manner similar to that for the prior art computers.

In one embodiment, a logical structure is imposed on memory devices 102-105 so that memory devices can be used in a manner functionally equivalent to prior art disk drives. Specifically, in this embodiment, each non-volatile memory device 104D, as illustrated in Figure 6, has a region corresponding to disk BOOT sector 104D-1, disk file allocation table (FAT) 104D-2, disk root directory 104D-3 and disk data sectors 104D-4. The specific configuration of the memory device depends upon the disk structure being emulated. In one embodiment, the following logical structure is used:

- 512 byte sectors;
- two copies of the FAT;
- 8 sectors per track;
- single sided disk;
- variable number of root directory sectors (function of memory size); and
- variable number of data area sectors (function of memory size).

Table 1 lists the memory, the FAT, the root directory size, and the characteristics required to configure a memory card so that the card has a logical structure.
equivalent to a disk. The size of the memory card is given across the top of Table 1 and the characteristics for a given size card are a column under the card size. In addition to the characteristics given in Table 1, the total number of sectors per directory is the bytes/directory divided by the number of bytes per sector, and the number of paragraphs is the bytes/card divided by 16. In this embodiment, the boot sector is the same size as other sectors. The data region of the memory is the bytes/card minus the sum of the bytes in the FAT, the root directory and the boot sector.

**TABLE 1**

<table>
<thead>
<tr>
<th>Card Size (bytes)</th>
<th>64K</th>
<th>128K</th>
<th>512K</th>
<th>1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 Sectors per track</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Number of sectors per FAT</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Number of Root Directory Entries</td>
<td>32</td>
<td>64</td>
<td>208</td>
<td>224</td>
</tr>
<tr>
<td>20 Bytes/Directory</td>
<td>32*32</td>
<td>32*64</td>
<td>32*208</td>
<td>32*224</td>
</tr>
</tbody>
</table>

The addressing of memory using the head, track and sector specified in the I/O mapped command can be done in a number of ways. For example, in one embodiment, the data area of the memory card is configured such that head one, track one, sectors 1-8 are in the first 4 Kbytes of the data area, head one, track two, sectors 1-8 in the next 4 Kbytes and so forth until the number of tracks being emulated is reached. If necessary, the structure is repeated for the second through the nth heads.

Accordingly, as explained more completely below, the location of any particular sector is simply reduced to calculating the memory address for the sector based upon the head, track and sector specified in the I/O command.

When memory cards 104, 105 are ROM cards, the ROM cards are configured either as a formatted disk (Fig. 6), as described above, or executable ROM code (Fig. 7A), or
combinations of a formatted disk and executable ROM code (Fig. 7B). One logical structure for ROM executable files, according to the principles of this invention, is given in Table 2 and Table 3 below and illustrated in Figure 7A. In this embodiment, if binary image files of any nature, in particularly directly executable code, are included on a ROM card, the ROM card has (i) a main header 104E-1 (Fig. 7A) (ii) at least one file header block 104E-2 and an executable image file 104E-3. The main header is used to identify the number of executable images in the ROM card, the specific card, and other information that characterizes the ROM card. The file header identifies an executable image file in the ROM and other information needed to load and execute the executable image file.

<table>
<thead>
<tr>
<th>Information</th>
<th>Length (bytes)</th>
<th>Data bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM Card Signature</td>
<td>2</td>
<td>4Ch, 59h</td>
</tr>
<tr>
<td>Length of card in 64k blocks</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>Signature ID</td>
<td>8</td>
<td>'ROMXXXXX0'</td>
</tr>
<tr>
<td>Number of File Header Blocks</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>ROM Header Version - MAJOR</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>- MINOR</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Reserved</td>
<td>15</td>
<td>15DUP(0)</td>
</tr>
<tr>
<td>ROM Checksum</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>Total Length</td>
<td>32 bytes</td>
<td></td>
</tr>
</tbody>
</table>

In Table 2, the first column lists the information that is stored in the main header. The second column gives length assigned to each piece of information and the third column gives specific information that is defined. Under "Reserved" the term "15DUP(0)" means duplicate zero fifteen times. Notice that in Table 2 the length of card
in 64 Kbyte blocks, the number of file header blocks, and the ROM checksum are listed as question marks. These items will vary depending upon the size of the memory on the card, and the number of executable image files on the ROM.

TABLE 3

<table>
<thead>
<tr>
<th>Information</th>
<th>Length (bytes)</th>
<th>Data bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>8</td>
<td>e.g. &quot;POQETROM&quot;</td>
</tr>
<tr>
<td>Name Extension</td>
<td>3</td>
<td>e.g. &quot;EXE&quot;</td>
</tr>
<tr>
<td>MAIN entry point offset</td>
<td>4</td>
<td>e.g. xxxx:yyyy</td>
</tr>
<tr>
<td>Reserved</td>
<td>17</td>
<td>17 DUP (0)</td>
</tr>
<tr>
<td>Total Length</td>
<td>32 bytes</td>
<td></td>
</tr>
</tbody>
</table>

In Table 3, the file header block (FHB) name is the name of the executable program. Name extension is the type of file. Thus, for example, an executable image of the Lotus 123 application would have the name "LOTUS123" and the name extension "EXE" to signify that the file is an executable image of Lotus 123. Lotus 123 is a spreadsheet user application available from Lotus Development Corp. of Massachusetts.

The main entry point offset in Table 3 is the double word at offset 0Bh from the start of the relevant file block header. The main entry point offset points to the offset of the entry point of main code from the start of free card space, i.e., after the main header and FHB.

If a ROM memory card is a mixed mode card 104M (Fig. 7B), i.e., the card contains both executable image files 104M-8 and formatted files 104M-4, the start of either logical structure must lie on a segment (64K) boundary. In one embodiment, the disk structure, i.e., the formatted files 104M-4, is at offset 0 from the start of the ROM card and main header 104M-6 for executable image files 104M-8 follows the ROM disk structure 104M-1 to 104M-4 and starts on a 64 Kbyte segment boundary 104M-5.

While the use of memories 102-105 (Fig. 4) provide
power, size and speed advantages over the prior art secondary memory, which typically consisted of floppy disks and hard disks, memories 102-105 are not I/O mapped even though the logical structure, described above, is imposed on the memories. Therefore, memories 102-105 are not directly usable with operating systems, such as MS-DOS, that are designed for I/O mapped secondary memory.

Nevertheless, according to the principles of this invention, I/O executive 109 (Fig. 4) accepts the I/O mapped commands from the operating system, changes the I/O mapping to a memory based mapping and performs the appropriate operations using one of memories 102-105. Hence, the large base of application programs for operating systems that use I/O mapped secondary memory can be used with computer 100.

In fact, I/O executive 109 provides functions equivalent to all the normal disk functions typically found in a prior art computer. In addition to the translation of the I/O mapped system commands to memory mapped commands, the computer of this invention also includes a memory mapping executive 108 so that memories 102-105 can also be used as expanded memory for computer 100. Therefore, according to the principles of this invention, each secondary memory device serves as a dual purpose device. In the prior art, two different devices were used to provide this functionality.

A flow diagram for processing user commands for disk operations, according to the principles of this invention, is illustrated in Figure 8. User commands for disk operations include, for example, reading data from a disk and writing data to a disk. In Figure 8, disk command 300, decompose command 301 and disk services 302 are included in operating system 350. Disk command 300 receives a user command for a disk operation such as formatting a disk, reading a file from disk, or writing a file to disk. Here, the user may be a user of computer 100 (Fig. 4) or a program operating in memory 101 of
computer 100 that performs disk operations. The user command from disk command 300 (Fig. 8) is passed to decompose command 301.

Decompose command 301 reduces the user command into a basic set of operations that are supported by the operating system. For example, if a file is being copied from one disk to another disk, decompose breaks the command into a open request for the file on the first disk, a copy command from the first disk to user memory, a close of the first file, an open of the second file, a write from main memory to the second disk and subsequently a closing of the second file. This sequence of operations is provided to disk services 302. The operations of disk command 300, decompose command 301 and disk services 302 are performed by operating system 350 and are known to those skilled in the art.

Disk services 302 is typically a device driver that receives the individual commands and provides an interrupt command which is sent to I/O executive 109. I/O executive 109 receives the disk services interrupt and converts the I/O mapped information in the interrupt to data that is sufficient to retrieve the requested information from hardware 303 or copy the requested information to hardware 303. In Fig. 8, hardware 303 includes main memory 101 and memory based peripheral storage units 303-1. Disk services 302, i.e. the disk service interrupt, and I/O executive 109 are described more completely below.

Figure 9 illustrates an example of a disk operation performed by I/O executive 109 using either credit card memory 104 (Fig. 3) or credit card memory 105 as peripheral storage unit 303-1. In this example, a read command for a file on credit card memory 104 has been issued in disk command 300 (Fig. 8). The operating system determines that the file is in a single sector 104-S (Fig. 9) and that the file is to be put at address xxxxx in user memory area 125. Decompose command 301 and disk services 302 generate the head, track, sector and data
buffer for reading the disk sector, i.e. the I/O mapping, and present this information in the form of an interrupt to I/O executive 109. These operations are equivalent to those that are performed in the prior art computers.

I/O executive 109 converts the I/O mapped request into a memory location which is passed to the hardware and the data in the appropriate segment of peripheral device memory is mapped from the peripheral device memory to main memory 101 by I/O executive 109. In another embodiment, the memory location generated by I/O executive 109 is passed to memory mapper 110 which in turn performs the mapping from the peripheral device memory to main memory 101. After completing the mapping, memory mapper 110 returns control to I/O executive 109. In contrast to the prior art nonvolatile RAM disks that were coupled to the operating system through a disk controller, I/O executive 109, in one embodiment, is a part of the ROM BIOS (Fig. 5) stored in main memory 101 of computer 100.

In Figure 9, the head, track and sector information for a floppy disk has been translated by I/O executive 109 and the desired sector 104-S has been identified as being located in the 64 Kbyte segment 104-SEG, which begins at address 20000h in memory card 104. In addition, the offset 104-OF from address 20000h has been calculated to specifically identify the 512 byte sector that has been requested by disk services 302 (Fig 8).

The 64 Kbyte segment starting at address 20000h (Fig. 9) is mapped by I/O executive 109 to the 64 Kbyte segment starting at address C0000h in system memory 126 of main memory 101. After the mapping from memory card 104 to system area 126 of main memory 101, I/O executive 109 copies sector 104-S from system area 126 into the appropriate position in user area 125. As explained more completely below, there are several additional operations which are performed to maintain the integrity of memory card 104 and system area 126, but the fundamental operations are as illustrated in Figure 9.
Figure 10 illustrates the process of memory mapping according to the principals of this invention using the memory in memory based peripheral storage units 102-105 (Fig. 4). EMS memory handler 310 (Fig. 10), typically a device driver in operating system 350, receives from a user application 355 executing in CPU 107 a request for information in a certain segment of memory on one of the four memory units 102-105 (Fig. 4), i.e., memory based peripheral storage units 303-1 (Fig. 10).

This request is reduced by memory services 311 to an expanded memory interrupt according to the LIM EMS 4.0 specification. The interrupt is processed by memory mapping executive 108 using memory mapper 110, as explained more completely below.

Memory mapping executive 108 maps the appropriate region of memory in units 303-1 into system memory area 126. Memory mapping from a peripheral storage unit is shown graphically in Figure 11. The data in memory segment extending from address X0000h to address X0000h + 64K of a peripheral storage unit is mapped into the segment of system area 126 starting at address D0000h.

While this mapping process accepts the prior art EMS LIM 4.0 specification interrupt as an input command, this mapping process is unique because the expanded memory is not memory on an expansion board as in the prior art, but rather memory in one of the disk drive locations of the computer. The prior art system addressed memory in one of the expansion slots. Accordingly, mapping information from peripheral storage devices to main memory achieves a new capability for such devices.

In one embodiment, disk services 302 of Figure 8 is the MS-DOS interrupt 13h:disk services. The interrupt 13h:disk services provides commands for get disk, read disk sectors, write disk sectors, verify disk sectors, format disk tracks, and get current drive parameters. Interrupt 13h has numerous other commands, but these commands are not supported in this embodiment of my
invention.

Table 4 defines interrupt 13h:disk services which are supported by I/O executive 109. In view of this disclosure, one skilled in the art can add support for 5 additional interrupt 13h:disk services to I/O executive 109. The calls in Table 4 are the information provided to I/O executive 109 of this invention and the returns comprise the information that is returned to the operating system by I/O executive 109.

TABLE 4

INTERRUPT 13H:DISK SERVICES

Function 00H:Reset Disk System

To call: 

<table>
<thead>
<tr>
<th>AH</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>CF = 0 no error</td>
</tr>
</tbody>
</table>

AL = drive number:

<table>
<thead>
<tr>
<th>AL</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>00-7Fh floppy disk</td>
<td>AH = error code (see Interrupt 13h Function 01h)</td>
</tr>
<tr>
<td>80-FFh fixed disk</td>
<td></td>
</tr>
</tbody>
</table>

Function 01h:Get Disk Status

To call: 

<table>
<thead>
<tr>
<th>AH</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>01h</td>
<td>AH = disk status:</td>
</tr>
<tr>
<td>00h</td>
<td>no error</td>
</tr>
<tr>
<td>01h</td>
<td>invalid command</td>
</tr>
<tr>
<td>02h</td>
<td>address mark not found</td>
</tr>
<tr>
<td>03h</td>
<td>write attempt on write-protected disk (F)</td>
</tr>
<tr>
<td>04h</td>
<td>sector not found</td>
</tr>
<tr>
<td>05h</td>
<td>reset failed (H)</td>
</tr>
<tr>
<td>06h</td>
<td>floppy disk removed (F)</td>
</tr>
<tr>
<td>07h</td>
<td>bad parameter table (H)</td>
</tr>
<tr>
<td>08h</td>
<td>DMA overflow (F)</td>
</tr>
<tr>
<td>09h</td>
<td>DMA crossed 64 KB boundary</td>
</tr>
<tr>
<td>0Ah</td>
<td>bad sector flag (H)</td>
</tr>
<tr>
<td>10h</td>
<td>data error</td>
</tr>
<tr>
<td>11h</td>
<td>ECC data error (H)</td>
</tr>
<tr>
<td>20h</td>
<td>controller failed</td>
</tr>
</tbody>
</table>
byte 3  sector-size code:
00h   128 bytes per sector
01h   256 bytes per sector
02h   512 bytes per sector (standard)
03h   1024 bytes per sector

As used herein, the register names, e.g., AX
(sometimes used as two registers called AH and AL), BX
(sometimes used as two registers called BH and BL), CX
(sometimes used as two registers called CH and CL), DX
(sometimes used as two registers called DH and DL), SP,
BP, SI, DI, CS, DS, SS, and ES, are those associated with
the Intel Corporation of Santa Clara, CA, iAPX86, 88 and
iAPX186, 188, 286, 386 and 486 family of microprocessor
systems and the interrupts are those associated with the
IBM PC ROM BIOS definition. However, these examples are
illustrative only of the principles of the invention and
are not intended to limit the scope of the invention to
the specific embodiment described. In view of this
disclosure, those skilled in the art can use the
information management executive of this invention with
other microprocessors and other operating systems.

The interrupt 13h commands supported by I/O executive
109 of this invention include reset system, get system
status, read sectors, write sectors, verify sectors, and
format track. Notice that data provided in interrupt
13h:disk services in the AH register corresponds to the
data required by I/O executive 109 in the AH register.
While this identity is not necessary, the identity
eliminates shuffling of the interrupt 13h registers prior
to invoking I/O executive 109.

To respond as a disk to the operating system, I/O
executive 109 must have information about certain charac-
teristics of the disk or disks being emulated. The
characteristics are included in a disk base table. One
embodiment of the disk base table is presented in Table 5.
- 29 -

40h seek failed
80h time out
AAh drive not ready (H)
BBh undefined error (H)
CCh write fault (H)
E0h status error (H)

Note:  H = fixed disk only,  F = floppy disk only.

Function 02h: Read Disk Sectors
Function 03h: Write Disk Sectors
10 Function 04h: Verify Disk Sectors
Function 05h: Format Disk Tracks

To call:

AH   = 02h  read disk sectors
     = 03h  write disk sectors
     = 04h  verify disk sectors
     = 05h  format disk track

AL   = number of sectors
CH   = cylinder number
CL   = sector number

DH   = head number
DL   = drive number
ES:BX = buffer address
       (unused if AH=05H)

25 Returns:

CF   = 0  no error
      = 1  error

AH   = error code (see Interrupt 13h Function 01h)

If AH was 05h on call:

30  ES:BX = 4-byte address field entries, 1 per sector:
    byte 0  cylinder number
    byte 1  head number
    byte 2  sector number
operating system so that effectively no operations are required when the operating system sends the reset command. Accordingly, I/O executive 109, in response to the reset command, sets the disk status bit in lower memory and returns control to the operating system. The location in lower memory and value of the disk status bit are known to those skilled in the art. (See for example, P. Norton and R. Wilton, The New Peter Norton Programmer's Guide to The IBM PC & PS/2, Microsoft Press, (1988).)

Similarly, when the operating system requests the disk system status by setting the AH register to 01h in interrupt 13h, I/O executive 109 retrieves the disk status bit, which has been previously set by another function, from the lower memory and places the status bit in the AH register. Control is then returned to the operating system.

When interrupt 13h has the AH register set to either 02h for reading sectors or 03h for writing sectors, as previously described, I/O executive 109 converts the geometrical information provided in interrupt 13h into a memory mapped address and then performs the operations requested. The steps performed by I/O executive 109 in response to either a read request or a write request are illustrated as a block diagram in Fig. 12. The general steps are the same for both requests, but the operations performed within a step may differ depending upon the request.

For interrupt 13h, the operating system sets the AH register and the other registers to the values given in Table 4. Receive interrupt 109-1 of I/O executive 109 receives these registers and determines the function to be performed, in this example either a read or a write. Processing then transfers to save configuration 109-2 which saves the CPU register configuration. Subsequently, choose segment 109-3 determines, as described more completely below, the memory area in system area 126 of main memory 101 to use in the read/write operation, and
TABLE 5

DISK BASE TABLE

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step-rate time and head unload time</td>
<td>0CFh</td>
</tr>
<tr>
<td>Head-load time</td>
<td>2</td>
</tr>
<tr>
<td>Wait time to motor turn-off</td>
<td>25h</td>
</tr>
<tr>
<td>Bytes per sector code: 2=512</td>
<td>2</td>
</tr>
<tr>
<td>Last sector number</td>
<td>8</td>
</tr>
<tr>
<td>Gap length between sectors</td>
<td>2Ah</td>
</tr>
<tr>
<td>Data transfer length</td>
<td>OFFh</td>
</tr>
<tr>
<td>Gap between sectors for track formatting</td>
<td>50h</td>
</tr>
<tr>
<td>Format data value</td>
<td>OF6h</td>
</tr>
<tr>
<td>Head-settle time</td>
<td>19h</td>
</tr>
<tr>
<td>Motor start-up time</td>
<td>4</td>
</tr>
</tbody>
</table>

The information of interest for this invention in the disk base table are the bytes per sector code, the last sector number and the format data value. Nevertheless, other information is included to ensure compatibility with the IBM XT computer architecture.

The read and write operations of I/O executive 109 are the most important functions of executive 109 because these functions perform the majority of disk services. Nevertheless, each of the functions performed by I/O executive 109 is discussed in turn.

The first interrupt 13h function serviced by I/O executive 109 of this invention is reset system wherein the AH register is set to 00h and the interrupt 13h invoked. This function is typically passed to I/O executive 109 when system 100 (Fig. 4) is rebooted. In an I/O mapped peripheral mass storage unit, such as a disk drive, the reset typically moves the head to the track zero position on the disk so that the system is ready to respond to further operating system commands. However, since the peripheral storage devices 102-105 (Fig. 4) of this invention are memory units, the devices are immediately ready to respond to further commands from the
then maps the 64 Kbyte segment containing the sector identified in interrupt 13h from the peripheral memory device to the specified segment in system area 126 of main memory 101.

Subsequently, analyze disk 109-4 checks to determine whether an error was found in choose segment 109-3, e.g., the specified device was not found. If an error was detected by choose segment 109-3, analyze disk 109-4 sets the appropriate error code and processing transfers to restore 109-8 which is described more completely below. If no error was found in choose segment 109-3, analyze disk 109-4 provides the address of the disk base table (Table 5) to valid request 109-5.

Valid request 109-5 compares information in disk base table (see Table 5) with information passed in interrupt 13h to determine whether the read/write request is valid. The specific operations performed in valid request 109-5, as described below, depend upon whether interrupt 13h specified a read or a write. If the read/write request is valid, processing transfers to locate sector 109-6. Conversely, if the read/write request is invalid, processing transfers to restore 109-8, which is described below.

Locate sector 109-6 calculates the offset of the sector, as described below, from the lower segment boundary and the total amount of data to be read/written. Subsequently, as explained more completely below, data transfer 109-7 copies the data specified in interrupt 13h from system area 126 to user area 125 for a read or from user area 125 to system area 126 for a write.

After data transfer 109-7 completes processing, restore 109-8 maps the 64 Kbyte segment from system area 126 to the appropriate peripheral storage unit, sets the disk status bit in lower memory, and restores the CPU configuration that was saved in save configuration 109-2. As noted above, when an error is detected, an error flag is set and processing transfers immediately to restore
109-8. In this case, the error flag is converted to an error code in addition to the other operations of restore 109-8. Return 109-9 returns the register values to interrupt 13h required to describe the operation performed by I/O executive 109.

The operations in choose segment 109-3 are shown in greater detail in Fig. 13. Choose segment 109-3 is used not only in the read function and the write function but also in the format function and the verify sector function, described below. In choose segment 109-3, after the AX and DX registers are pushed onto the stack by save configuration 109-30, the next operation is locate drive 109-31. Locate drive 109-31 ascertains whether the internal drive C, i.e., ROM 102 (Fig. 4) has been requested, or internal drive D has been requested and if either drive was requested processing transfers to select segment 109-34. Conversely, if locate drive 109-31 ascertains that either the external A or B drive has been specified in the read or other request, processing transfers to device present 109-32.

Device present 109-32 checks the specified external drive location to determine whether a memory card is inserted in the drive. In one embodiment, one of the memory card pins causes the voltage level on a line to change which in turn results in a bit in an I/O port being changed. Device present 109-32 checks this bit to ascertain whether a device is loaded in the drive. If the specified drive contains a memory card, processing transfers to select segment 109-34 and otherwise an error flag is set by error return 109-33 and processing returns to the function that called choose segment 109-3.

Select segment 109-34 initializes either the 64 Kbyte segment of system memory starting at address C0000 or the 64 Kbyte segment of system memory 126 (Fig. 4) starting at address D0000. Processing transfers from select segment 109-34 to calculate segment 109-35 which identifies the 64 Kbyte segment in the specified memory device that contains
the sectors that have been specified in the interrupt call from the operating system. Recall that, as shown in Figs. 6 and 7B, the memory cards of this invention have been formatted so that the head number, track number and sector number provided in interrupt 13h are sufficient to locate the appropriate 64 Kbyte memory segment on the selected memory card. Accordingly, the logical segment number on the selected memory device is calculated using the head, track, and sector number.

For example, consider a memory emulating a double-sided floppy disk having 40 tracks per side and 8 sectors per track. Assume that the memory was configured so that the data on the first side of the disk is contained in consecutive positions in the memory followed by the data on the second side of the disk. If the request to choose segment 109-3 specified head 2, i.e., the second side of the disk, track 6, sector 4, the memory address must first be offset by the memory space used by side 1 of the disk which is 40 tracks times 8 sectors per track or 320 sectors. Second, the distance in the memory corresponding to the first five tracks on the second side is 5 tracks times 8 sectors per track or 40 sectors. The sum of the sectors on the first side of the disk and the sectors in the first five tracks on the second side of the disk plus the number of the sector in track 6, i.e. 4, is the logical sector number, i.e. 324.

To find the logical segment number, i.e. the 64 Kbyte segment containing logical sector number 324, the logical sector number is multiplied by the number of bytes per sector, typically 512 bytes, and divided by the number of bytes per logical segment, i.e., 64 Kbytes, to generate the segment locator. The modulo of the segment locator is the logical segment number which contains the sector requested by the interrupt from the operating system. The general expression for the logical segment number in this embodiment is given in Equation 1.

\[
Segment \ Locator = \frac{512 \times \text{logical sector number}}{64 \times 1024}
\] (1)
Logical Segment Number = Modulo (Segment Locator)

After calculate segment 109-35 determines the logical segment number, map logical segment 109-36 copies the identified 64 Kbyte segment from the indicated memory card to the segment in system area 126 identified by select segment 109-34. Subsequently, restore configuration 109-37 pops the AX and DX register values from the stack and return 109-38 returns processing to the I/O executive function that called choose segment 109-3.

Fig. 12 illustrates the general steps for a read or write operation by the I/O executive 109 of this invention. However, as described more completely below, some of these steps have different operations within the general steps. In the read operation, as shown in Table 4, interrupt 13h from the operating system provides the number of sectors to be read, the track number, the starting sector on the track, the head number, the drive number, and the address of the memory buffer.

In the read operation, receive interrupt 109-1, save configuration 109-2, choose segment 109-3 and analyze disk 109-4 function as described above. Valid request 109-5 uses the disk base table (Table 5) to ascertain whether the requested sector is contained on the disk, e.g., was sector 10 requested for a disk having only eight sectors per track. If the sector request is valid, processing transfers to locate sector 109-6. Conversely, if the sector request is not valid, processing transfers to restore 109-8 as described previously.

Locate sector 109-6 subtracts the logical segment number (Equation 1) from the segment locator to generate offset 104-OF (Fig. 9) of sector 104-S within the 64 Kbyte segment which is now in system area 126 of main memory 101. The number of sectors to be read is then obtained from the data provided by interrupt 13h and the total number of bytes that must be transferred from system area 126 to user area 125 is calculated, i.e., the number of
sectors to be read times 512 bytes per sector. Thus, the offset, and the total number of bytes to be copied are known and processing transfers to data transfer 109-7 (Fig. 12). Data transfer 109-7 copies the requested data from system area 126 (Fig. 9) into user area 125 specified by the address in the buffer provided in the interrupt command. Note that if the number of sectors is greater than 64K bytes, the operating system issues a number of sequential I/O commands.

The information passed to I/O executive 109 of this invention by interrupt 13h for a write operation is nearly identical to the information passed for a read operation except the AH register is set to three. (See Table 4) Accordingly, receive interrupt 109-1 (Fig. 12), save configuration 109-2, choose segment 109-3, and analyze disk 109-4 again function as previously described. However, valid request 109-5 performs some other operations in addition to those described above for a read operation.

Valid request 109-5 first checks to determine whether there is sufficient space in the specified memory device to write the number of sectors specified in interrupt 13h. The available sectors in the memory device are compared with the requested sectors to ascertain whether sufficient space is available. The number of bytes per sector is obtained from the disk base table for subsequent processing. Subsequently, valid request 109-5 checks the specified memory device to ascertain whether the device is write protected.

In one embodiment, a read, write, read, compare sequence is used to ascertain whether the memory device is write protected. A first byte is read from a specified memory address, the byte is incremented so that the new byte is different from the byte read from memory. The new byte is written to the specified memory address and a second read is made at the specified memory address. The byte obtained from the second read is compared with the
incremented byte that was written to memory. If the bytes are the same, the memory is not write protected, but if the bytes are different, the memory is write protected. In another embodiment, the memory card could have an output pin that had a first state, i.e., a voltage level, if the card was write protected a second state, different from the first state, if the card was not write protected. For this card, valid request 109-5 would check the state of the specified output pin for the memory card. If the specified memory device is write protected, processing transfers to restore 109-8.

If the specified memory device is not write protected, processing goes to locate sector 109-6. The bytes per sector and the number of sectors are used in locate sector 109-6 to calculate the total number of bytes to transfer. The offset of the sector in the 64 Kbyte segment in system area is calculated as described above. Since interrupt 13h provided the address of the data in main memory, the number of bytes to be moved from main memory into the system area memory and the offset of the sector in system area 126 are known. Processing transfers to data transfer 109-7 which copies data from user area 125 of main memory 101 into system area 126. The remaining operations are equivalent to those previously described with respect to Fig. 12.

In one embodiment, I/O executive 109 in response to a verify request, i.e. interrupt 13h function 4, simply sets the appropriate registers to indicate success and returns control to the operating system.

The final operation supported by I/O executive 109 of this invention is the formatting of a memory card. In one embodiment, a computer program formats the memory device by building the boot sector, FAT, and the root directory, as described above. Specifically, the data required for a standard boot sector is loaded from the program into the boot section location 104D-1 (Fig. 6) Similarly, the program initializes the locations in FAT 104D-2 and
directory 104D-3 to the values defined by the MS-DOS operating system. The data and the format of the data that is initialized are known to ones skilled in the art.

The format program then initiates an interrupt 13h calling the format function. The format function in I/O executive 109 performs receive interrupt 109-1 (Fig. 12), save configuration 109-2, choose segment 109-3, and analyze disk 109-4, as described above. In valid disk 109-5, the format function checks the head number, the number of bytes per track, and whether there is enough space for the requested format. The disk base table is used to obtain the number of bytes per track, the format data value and then the specified memory device is checked using the read, write, read, compare sequence, described above, to determine whether the device is write protected.

If the device is write protected, the disk status bit in lower memory is set and processing returned to the calling program. If the device is not write protected, the format data value from the disk base table is written in the specified sectors and then the processing transfers to restore 109-8 (Fig. 12), which subsequently returns control to the format program.

In the embodiment shown in Fig. 13, choose segment 109-3 of I/O executive 109 performed the necessary page swapping operations between the memory card and system area 126 of main memory 101. However, if the hardware associated with I/O executive 109 changes, the page swapping features of I/O executive 109 in response to interrupt 13h must be changed. Accordingly, in another embodiment, the page swapping is not performed by I/O executive 109 but rather an interrupt 66h in memory mapper 110 is called by I/O executive 109 in choose segment 109-3. Interrupt 66h performs the memory swapping and then returns control to I/O executive 109. This approach has the advantage that if the hardware changes, the changes required to I/O executive 109 are minimized because the necessary changes are made in interrupt 66h of
memory mapper 110.

To call interrupt 66h, I/O executive 109 in choose
segment 109-3 initializes the registers as shown in
Table 6 and issues an interrupt 66h.

TABLE 6

Interrupt 66h:Function 02h    Hardware Control.
Service 00h    Page swapping control.

Parameters Passed:

AH     Function number (02h)

AL     Service number (00h)

CX     Page number (16Kb pages)

DL     Device 0 - External memory card 1

        (Drive A)

        1 - External memory card 2

        (Drive B)

        2 - Internal ROM (Drive C)

        3 - Internal RAM/ROM (Drive D)

        1 - Unswap page segment

DH     Swap page segment address in 16kb blocks

        starting at C0000h.

        00 - C000  08 - E000
        01 - C400  09 - E400
        02 - C800  0A - E800
        03 - CC00  0B - EC00

        04 - D000  0C - F000
        05 - D400  0D - F400
        06 - D800  0E - F800
        07 - DC00  0F - FC00

Parameters returned:

CF     Error flag

AX     Error code

-3     Parameter error
-5     Invalid device
-6     Page out of bounds
-7     Illegal swap segment
-8     16k block not supported
-9     External Memory card not present
In one embodiment, interrupt 66h contains a number of functions with each function including a number of services. Accordingly, interrupt 66h is specified with function 2, which, in this embodiment, is the hardware control function that supports page swapping control services as service zero. The hardware control function 2 also includes other services, for example, tone generation, as service 1, get current video controller as service 2, and set current video controller as service 3, and page swapping status as service 4. However, for the embodiment described herein, service 0, page swapping control, is of primary importance, and service 4, page swapping status is available for providing status information if that information is needed. The other services are unrelated to peripheral device information management and so are not discussed further. For a complete description of one embodiment of interrupt 66h, see copending, commonly assigned, and commonly filed application of John P. Fairbanks, et al., entitled "Portable Low Power Computer", U.S. Serial No. 07/375,721, which is incorporated herein by reference.

When memory mapper 110 is invoked by a calling function, e.g., choose segment 109-3, interrupt 66h first saves the CPU register configuration and then checks to determine the function requested. Control subsequently is passed to the requested function which in turn ascertains the service specified. Accordingly, in this embodiment, interrupt 66h passes control to function two, and function two performs a check service request to determine the service specified in the interrupt command. When service request check determines that service 0, page swapping control, is requested, processing progresses as illustrated in the flow diagram of Fig. 14.

Page number check 110-1 first tests to determine whether the requested memory swap is a swap of a 64 Kbyte segment or a swap of some other size segment. If a 64 Kbyte was specified, processing transfers to maximum
size check 110-2, otherwise processing transfers to error return 110-3. Error return 110-3 sets the error code, restores the saved configuration and returns control to the function that sent interrupt 66h.

5 Maximum size check 110-2 determines whether the 64 Kbyte segment requested is within the maximum size of the memory cards. For example, if the maximum size memory card is a 16 megabyte card and the segment request was for 400h, 400h corresponds to the 257th 64 Kbyte segment, which is greater than 16 megabytes. Accordingly, in this example, maximum size check 110-2 would terminate processing and transfer control to error return 110-3. Maximum size check 110-2 is included in interrupt 66h to maintain the LIM EMS 4.0 specification, as described more completely below.

If the requested segment is within the physical boundary of the maximum memory card size, check device 110-4 ascertains whether the device requested for the page swap is a valid device, i.e., is the device specified in interrupt 66h one for which memory swapping is allowed. If an error is not detected by check device 110-4, processing transfers to memory segment check 110-5.

As previously described, a 64 Kbyte segment is swapped between a peripheral storage device and main memory. Further, in this embodiment, the 64 Kbyte segment must start at a 64 Kbyte segment boundary. Therefore, memory segment check 110-5 determines whether the first page specified for the memory swap starts at a 64 Kbyte segment boundary. If the first page starts on a segment boundary, memory segment check 110-5 is successful, but if the page starts within a 64 Kbyte segment, processing transfers to error return 110-3.

After successful completion of memory segment check 110-5, unswap page check 110-6 ascertains whether a page is being mapped from a memory device into system memory, i.e., a swap or whether information is being mapped from system memory to a memory device, i.e. an unswap. If
processing specified by interrupt 66h is a swap, the initial system memory address for receiving the data from the specified memory device is determined. Subsequently, processing transfers to find device 110-7 which ascertains whether the device specified in interrupt 66h is an internal device, i.e., drives C and D or an external device, i.e., drives A and B. If an external device was specified, find device 110-7 determines whether a memory card is plugged into the appropriate drive.

Find device 110-7 subsequently transfers to map 110-8 which swaps the specified 64 Kbyte segment from the specified memory device into the 64 Kbyte segment in system memory area 126. After the swap is completed, processing transfers to restore 110-9 which updates the low memory BIOS pointer, returns the CPU registers to their original values, and returns to the calling function.

If interrupt 66h call specified an unswap, unswap page check 110-6 determines the 64 Kbyte segment in system memory 126 to unswap. Subsequently, unswap 110-11 defines the I/O port for unmapping. One embodiment of I/O ports suitable for use with interrupt 66h is disclosed in the copending, commonly assigned and commonly filed patent application of John P. Fairbanks, et al., entitled "Portable Low Power Computer", U.S. Serial No. 07/375,721, which is incorporated herein by reference. Then, unswap 110-11 transfers control to map 110-8 which maps the information from main memory 101 to the appropriate device and then transfers to restore 110-9, which in turn functions as described above.

In the embodiment of computer system 100, described above, computer system 100 executes user applications that permit temporary suspension of the application and initiation of another user application. Thus, interrupt 66h, function 2, service 4, which obtains information about the page swapping status, is provided so that information management executive 111 can be utilized in
either such a suspension and initiation or a multitasking environment. The purpose of this service is to determine whether another application is using a segment in system area 126 of main memory 101 or whether the segment is available for swapping.

In response to interrupt 66h, function 2, service 4, memory mapper 110 first determines whether the parameters passed in the interrupt are valid. For example, the page is checked to ascertain whether a valid region in memory was specified. Next, the segment in main memory that was requested is identified. Finally, a data area for memory mapper 110 is interrogated to ascertain whether the requested segment is being used. The status of the requested segment is returned to the calling program.

According to the principles of this invention, I/O executive 109 provides a computer architecture in which memory mapped devices are utilized in the same manner as prior art I/O mapped disks. Accordingly, I/O executive 109 provides a new capability not found in the prior art microcomputers. In particular, the access time to secondary memory is significantly enhanced because mass storage devices, having an access time on the same order magnitude as the main memory of the computer, can be accessed. The mass storage devices are suitable for use in portable battery powered computers. Finally, I/O executive 109 supports available operating systems so that user applications can be used without modification.

Another novel feature included in information management executive 111 (Fig. 4) of this invention is memory mapping executive 108. As previously described, memory cards 104, 105, ROM card 102, and RAM card 103 may be accessed as either I/O mapped devices, as described above, or memory mapped devices.

Memory mapping executive 108 provides systems and application developers a means to page data into and out of system area 126. Memory mapping executive 108 responds to interrupts 67h as defined in the LIM EMS 4.0
specification. The LIM EMS 4.0 interrupt 67h was used for interfacing with memory mapping executive 108 because this technique is familiar to developers and provides a uniform standard.

In one embodiment, memory mapping executive 108 would support functions 1, 2, 3, 4, 5, 6, 7, 12 and 13 in the LIM EMS 4.0 interrupt 67h. The various function calls for interrupt 67h that would be supported by memory mapping executive 108 and the information returned by memory mapping executive 108 upon completion of the function are given in Table 7. In this embodiment, the error codes generated by memory mapping executive 108 are given in Table 8. These definitions are equivalent to those for the functions in the LIM EMS 4.0 interrupt 67h.

<table>
<thead>
<tr>
<th>TABLE 7</th>
</tr>
</thead>
</table>

**Interrupt 67h Functions**

**Function 1:Get Status**

Passed: AH 40h  
Returns: AH Error code

**Function 2:Get Page Frame Address**

Passed: AH 41h  
Returns: AH Error code  
if AH = 0  
BX page frame segment address

**Function 3:Get Unallocated Page Count**

Passed: AH 42h  
Returns: AH Error code  
if AH = 0  
BX number of unallocated pages  
DX total number of pages

**Function 4:Allocate Pages**

Passed: AH 43h  
BX Number of pages needed  
Returns: AH Error code  
if AH = 0  
DX RMM handle (in the range 1 - FFh)
Function 5: Map/Unmap Handle Pages

Passed: AH 44h
AL Physical Page Number
BX Logical Page Number
DX RMM Handle
if ES:DI points to Device ID ‘RMMDEVXX’
CX Device identification:
0 Drive A:
1 Drive B:
2 Internal ROM otherwise extended memory is assumed

Returns: AH Error code

Function 6: Deallocate Pages

Passed: AH 45h
DX RMM Handle

Returns: AH Error code

Function 7: Get Version

Passed: AH 46h

Returns: AH Error code
AL version number
AL is returned as major/minor
BCD version number, e.g.
for v1.0:
0001 0000

Function 12: Get Handle Count

Passed: AH 4Bh

Returns: AH Error code
BX total number of open handles

Function 13: Get Handle Pages

Passed: AH 4Ch
DX RMM handle

Returns: AH Error code
BX number of pages allocated to handle
TABLE 8

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Successful</td>
</tr>
<tr>
<td>80h</td>
<td>Mapping executive software malfunction</td>
</tr>
<tr>
<td>81h</td>
<td>Mapping executive hardware malfunction</td>
</tr>
<tr>
<td>83h</td>
<td>Invalid handle</td>
</tr>
<tr>
<td>84h</td>
<td>Invalid function request</td>
</tr>
<tr>
<td>85h</td>
<td>All handles used</td>
</tr>
<tr>
<td>86h</td>
<td>Page mapping context error</td>
</tr>
<tr>
<td>87h</td>
<td>Not enough memory space in system</td>
</tr>
<tr>
<td>88h</td>
<td>Not enough unallocated pages</td>
</tr>
<tr>
<td>89h</td>
<td>Attempt to allocate zero pages</td>
</tr>
<tr>
<td>8Ah</td>
<td>Logical page out of range</td>
</tr>
<tr>
<td>8Bh</td>
<td>Physical page out of range</td>
</tr>
<tr>
<td>8Fh</td>
<td>Invalid parameter</td>
</tr>
</tbody>
</table>

Memory mapping executive 108 performs two distinct operations. The first operation is mapping information between a peripheral memory device 102-105 (Fig. 4) and main memory 101. The second operation is mapping information between expanded memory on an expanded memory board and main memory 101.

Computer system 100 in the embodiment shown in Figure 4 does not include means for connecting an expanded memory board into the system. Nevertheless, memory mapping executive 108 in computer system 100 may include expanded memory mapping capability thereby permitting expansion of computer system 100 to include an expanded memory board in other embodiments.

A first sequence of steps are used for memory mapping between peripheral memory devices 102-105 and main memory 101 and a second sequence of steps are used for memory mapping between an expanded memory board and main memory 101. The two sequence of steps are similar, except more steps are needed in the mapping between an expanded memory board and main memory 101.

The sequence of steps used for mapping information between peripheral devices 102-105 and main memory 101 are given in Table 9 while the steps that would be used for mapping between main memory 101 and an expanded memory board are given in Table 10 below.
TABLE 9

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Check if memory mapping executive is installed</td>
</tr>
<tr>
<td>2.</td>
<td>Get page frame address</td>
</tr>
<tr>
<td>3.</td>
<td>Map in pages from a device</td>
</tr>
<tr>
<td>4.</td>
<td>Read/write/execute in main memory</td>
</tr>
<tr>
<td>5.</td>
<td>Unmap pages from main memory.</td>
</tr>
</tbody>
</table>

The first step in Table 9 is to determine whether memory mapping executive 108 of this system is installed in the computer system. Since memory mapping executive 108 couples the EMS memory handler 310 and hardware 303, (Fig. 10) the memory mapping capability between memory based peripheral storage units 303-1 and main memory 101 is not available if memory mapping executive 108 is not installed in the computer system.

The second step is to obtain the page frame address in system area 126 of main memory 101. The page frame address is the main memory 101 address at which the swapped information is to be initially loaded. When the page frame address is known, sufficient information is available to map information from a peripheral memory device to main memory 101, i.e., step 3 of Table 9. After the information is mapped from the peripheral device to main memory 101, the desired operations, for example, reading or writing or executing in main memory, i.e., step 4 of Table 9, are performed.

When the use of the information is complete, the final step, Step 5, the pages are unmapped from main memory 101. The unmapping maps the information from main memory 101 back to the appropriate memory based peripheral storage unit 303-1. The precise functions of interrupt 67h which are used to perform this sequence of steps are described more completely below.

The sequence of steps used for accessing and using expanded memory is similar to those in Table 9, but more steps are required to maintain conformity between memory
mapping executive 108 and the LIM EMS 4.0 specification. The steps in this sequence are given in Table 10.

\textbf{TABLE 10}

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1. Check if memory mapping executive is installed</td>
</tr>
<tr>
<td></td>
<td>2. Determine if enough pages exist</td>
</tr>
<tr>
<td></td>
<td>3. Allocate logical pages</td>
</tr>
<tr>
<td></td>
<td>4. Get page frame address</td>
</tr>
<tr>
<td></td>
<td>5. Map in pages from expanded memory board</td>
</tr>
<tr>
<td>10</td>
<td>6. Read/write/execute memory</td>
</tr>
<tr>
<td></td>
<td>7. Unmap pages from main memory</td>
</tr>
<tr>
<td></td>
<td>8. Deallocate pages</td>
</tr>
</tbody>
</table>

The first step of Table 10 is also to determine whether memory mapping executive 108 of this invention is installed in the computer system. After memory mapping executive 108 is determined to be in the system, the next step is to determine whether sufficient logical pages exist in expanded memory for the memory swap between expanded memory and main memory 101.

When sufficient logical pages are available, the logical pages are allocated for the memory swap (Table 10, Step 3) as described below, and the page frame address obtained (Table 10, Step 4). When the page frame address is known, sufficient information is available to map information from the expanded memory board to main memory 101 and then read, write or execute in the selected segment of main memory 101. When the desired operations are complete, the pages are unswapped and deallocated, which completes the operation of the expanded memory.

One embodiment of memory mapping executive 108 includes functions to perform the steps in Table 9. In an alternative embodiment, the functions required to perform the steps in both Table 9 and Table 10 could be included in memory mapping executive 108.

In one embodiment, the smallest block of information that can be swapped in or out of system memory 126 is a full 64 Kbyte segment, i.e., four 16 Kbyte pages under the
LIM EMS 4.0 standard. Since memory mapping executive 108 adheres to the LIM EMS 4.0 standard, the page allocation must always be in multiples of four 16 Kbyte pages, i.e., a 64 Kbyte memory segment. The largest available contiguous block of main memory is generally 192 Kbytes or twelve 16 Kbyte pages. While in most applications only twelve pages are available, up to about sixteen pages are available if the main memory segment with addresses ranging from F0000h to FFFFFFFh is used. However, as shown in Fig. 5, this memory segment contains the ROM BIOS.

Thus, if this segment is used in a memory swap, care must be taken to assure that overwriting of the ROM BIOS does not affect operation of computer 100.

To test for the presence of memory mapping executive 108 in a computer, the contents of interrupt 67h vector location i.e., the information at address 0000:019C in lower memory, is obtained. This vector points to the start of the memory mapping executive entry code. The device name field ASCII string at offset 0Ah into the start of the ROM entry code is compared with 'RMMXXXXX0', and if the comparison is true, memory mapping executive 108 is available in the computer.

If memory mapping executive 108 is not available, executive 108 must be installed in the computer system.

Memory mapping executive 108 is installed in one embodiment as a terminate and stay resident program.

In another embodiment, which would support all the functions of Table 7, the installation of memory mapping executive 108 would create two counters and a table. The first counter, a handle counter, would be used to monitor the number of handles that are used by memory mapping executive 108. When information in memory is mapped from expanded memory into system area 126 of main memory 101, memory mapping executive assign each page of expanded memory used in the mapping the same number. The assigned number would be between one and 254. This number, the handle, would be used by memory mapping executive 108 to
identify the mapping in subsequent operations. A second counter, page count, would be used to monitor the number of logical pages that are used.

The table created on the installation of memory 5 mapping executive 108 would be the handle page table which is an array of bytes, one byte for each logical 16 Kbyte page. As used herein, a logical page refers to a portion of memory in one of the peripheral memory devices or in extended memory. The handle page table array would map each logical page to the appropriate handle.

Accordingly, after memory mapping executive 108 is installed and the check is performed as described above, the use of interrupt 67h is controlled by an application program as shown in Fig. 10. The sequence of interrupt 15 67h calls depends upon whether expanded memory on an expanded memory board or memory in one of the peripheral memory devices 102-105 is being mapped into system memory.

In one embodiment, only peripheral devices 102, 104, 20 105, are utilized in the memory mapping operations. In mapping memory from one of these devices to system area 126, user application program 355 first causes operating system 350 to issue an interrupt 67h function 2 to obtain the page frame address (Table 9, Step 2). The page frame 25 segment address in this embodiment must be either address C0000h, D0000h or E0000h. Typically, function 2 of memory mapping executive 108 sets the BX register to C0000h and returns processing to the calling program.

The operating system then issues an interrupt 67h 30 function 5 to map information in a specified peripheral memory based device into system area 126 of computer system 100 (Fig. 4). Interrupt 67h function 5 supplies memory management executive 108 with the physical page number, the logical page number and the memory device to be used for the mapping. To specify the peripheral storage device, the address ES:DI points to "RMMDEVXX" and the CX register contain a number identifying the device.
Specifically, CX is set to 0 for memory card 104 (Drive A), to 1 for memory card 105 (Drive B), and to 2 for memory card 102 (Drive C). If the address ES:DI does not point to "RMMDEVXX", memory mapping executive 108 uses expanded memory.

The first operation in memory mapping executive 108, in response to interrupt 67h function 5 is to save the CPU register configuration, as illustrated in Figure 15. Save configuration 108-10 (Fig. 15) pushes the registers onto the stack and sets the data segment register. Save configuration 108-10 passes processing to device check 108-11. Device check 108-11 compares the string at address ES:DI with "RMMDEVXX" to ascertain whether expanded memory or a memory based peripheral unit was specified. If expanded memory was specified and an expanded memory board is coupled to the computer system, processing passes to check handle 108-12. If expanded memory was specified and no expanded memory board is coupled to computer system 100, processing transfers to error return 108-20.

Conversely, if address ES:DI points to "RMMDEVXX", device check 108-11 ascertains whether one of the allowed peripheral devices was specified, i.e., Drive A, Drive B or the internal ROM card. Further, if either Drive A or Drive B was specified, device check 108-11 determines whether a memory card is plugged into the appropriate drive. If any of these tests detect an error, device check 108-11 sets an appropriate error flag and passes processing to error return 108-20. If the specified peripheral storage unit is available, processing branches over check handle 108-12 and logical page check 108-13 to mapping check 108-14. Check handle 108-12 and logical page check 108-13 are described below.

Mapping check 108-14 determines whether interrupt 67h function 5 requested mapping of a logical page from a peripheral device to the system area 126 of main memory or an unmapping from system area 126 to a peripheral memory
device. If an unmapping was requested, processing transfers to set register 108-15 which subsequently sets the DL register to a -1 so that the call to interrupt 66h, described below, results in an unmap. Set register 108-15 subsequently transfers processing to check physical page 108-16. If a mapping was specified in interrupt 67h function 5, processing transfers directly from mapping check 108-14 to check physical page 108-16.

Check physical page 108-16 compares the physical page number with the maximum physical page number permitted to ascertain whether the physical page is within the maximum number permitted. Again, if the physical page number is not within the specified bounds, an error flag is set and processing transfers to error return 108-20. Conversely, processing transfers to setup registers 108-17.

Set up registers 108-17 converts the number of pages to 64 Kbyte and places the device number in register DL, the page number in register CX, and the swap page segment in register DH. The AX register is set to 2000h. After set up registers 108-17, memory mapping executive 108 sends an interrupt 66h to memory mapper 110.

Interrupt 66h in memory mapper 110 was described above, and that description is incorporated herein by reference. Interrupt 66h swaps or unswaps the specified memory segment or segments from the peripheral memory to the specified area in system area 126 of main memory 101. After interrupt 66h finishes processing, control returns to error check 108-19 of memory mapping executive 108.

Error check 108-19 subsequently examines the values returned by interrupt 66h to determine whether an error was generated in interrupt 66h. If an error was detected, the hardware malfunction error flag is set and processing transfers to error return 108-20. If no error was detected, processing transfers to restore 108-21 which sets up the registers returned to interrupt 67h function 5 and pops the configuration saved on the stack back into the appropriate CPU registers.
The information swapped into the system area may be data which is addressed by a program in user area 125 or conversely may be an executable image of a file which is executed as described more completely below. After the use of the information swapped into system area 126 is complete, an interrupt 67h function 5 is sent to memory mapping executive 108 specifying that the pages are to be unswapped from system area 126. The unswap operation of memory mapping executive 108 is similar to the mapping operation described above except mapping check 108-14 passes control to set register 108-15, which functions as described above. Similarly, set up registers 108-17 configures the registers for an unswap so that interrupt 66h 108-18 maps the information from memory area 126 to the appropriate location in the peripheral storage device.

Memory mapping executive 108 of information management executive 111 provides a new computer architecture with significantly enhanced performed over prior art computers. Previously, memory mapping was limited to an expanded memory board, but memory mapping executive 108 swaps information between the same devices that are used for I/O mapped commands. Thus, according to the principles of this invention, a means for using the same device in two different modes of operation, i.e., I/O mapped and memory mapped, is provided.

In another embodiment, memory mapping executive 108 also supports the sequences of steps in Table 10 for mapping between expanded memory and main memory 101. The first step of checking whether memory mapping executive 108 is installed was described above and is incorporated herein by reference. The application program through EMS memory handler 310 performs step 2 of Table 10 by using interrupt 67h, function 3.

Interrupt 67h, function 3 of memory mapping executive 108, in response to the interrupt command, obtains the value of the logical page counter, described above, and subtracts this value from the maximum number of logical
pages in expanded memory and returns the difference as the unallocated page count. After determining the unallocated page count, function 3 restores the data segment register and places the appropriate values in the AH, BX and DX registers.

The application program through EMS memory handler 310 (Fig. 10) next sends an interrupt 67h function 4 to memory mapping executive for an allocation of logical pages. In response to interrupt 67h function 4, memory mapping executive 108, as illustrated in Fig. 16, first saves the contents of the BX and DS registers in save configuration 108-1 and then passes processing to check handles 108-2. Check handles 108-2 compares the maximum number of handles with the current value of the handle counter. If the value of the handle counter is less than the maximum number of handles, control is passed to page check 108-3. If the handle count is greater than the maximum number of handles, an error flag is set and processing transfers to restore 108-8.

Page check 108-3 determines whether, in the information passed by interrupt 67h function 4, a nonzero number of pages was requested. If the number of pages was zero, an error allocation flag is set and processing branches to restore 108-8. If a nonzero number of pages was requested, page check 108-3 compares the available pages with the requested pages to determine whether there is sufficient space in expanded memory for the allocation. If there is not sufficient space, a page error flag is set and processing transfers to restore 108-8.

Conversely, if sufficient pages are available, page check 108-3 (Fig. 16) then determines whether four 16 Kbyte pages have been requested. As described above, in this embodiment, the number of pages requested must be an integral multiple of 64 Kbyte. Accordingly, if the page request is not a 64 Kbyte block, a software error flag is set and processing transfers to restore 108-8.
However, if a nonzero number of pages was requested, sufficient pages were available for the request, and a 64 Kbyte block of pages was requested, processing transfers to increment handle count 108-4. Increment handle count 108-4 increments the handle counter. Get next handle 108-5 ascertains the value of the next handle available and transfers processing to update handle page table 108-6.

Update handle page table 108-6 adds the new handle number to handle page table and stores the handle with each logical page of expanded memory. Finally, processing transfers to update page count 108-7 which adds the number of logical pages allocated to the page counter, and sets the AH register to success. Subsequently processing transfers to restore 108-8. Restore 108-8 resets the registers pushed in save configuration 108-1 and sets an error flag if appropriate. Return 108-9 returns control to EMS handler 310.

After allocation of logical pages, the application program sends interrupt 67h function 2 to memory mapping executive 108 to obtain the page frame segment address. The operation of memory mapping executive in response to interrupt 67h function 2 was described above.

EMS memory management handler 310 next sends an interrupt 67h function 5 to memory mapping executive 108. As previously described interrupt 67h function 5 supplies the physical page number, the logical page number, and the handle associated with the mapping. Since in this embodiment, expanded memory is being used, the address ES:DI does not point to "RMMDEVXX".

The operation of memory mapping executive 108 in response to the interrupt 67h function 5 was described above. The operation in the embodiment is identical to the above description except device check 108-11 (Fig. 15) passes processing to check handle 108-12 because a peripheral memory device was not specified.
Check handle 108-12 compares the handle number
supplied in interrupt 67h function 5 with the handles in the handle page table. If the handle is valid, the entry in the handle page table for the handle is obtained so that the physical page number and the logical page number for the mapping are available. Conversely, if the handle is invalid, processing transfers to error return 108-20 which restores the configuration saved in save configuration and returns processing to EMS memory management handler 310 as previously described.

Logical physical page 108-13 compares the logical page number with the maximum logical page number permitted to ascertain whether the logical page is within the maximum number permitted. Again, if the logical page number is not within the specified bounds, an error flag is set and processing transfers to error return 108-20. Conversely, processing transfers to mapping check 108-14.

After logical page check 108-13, processing continues, as illustrated in Figure 15, and the information from expanded memory is swapped, sometimes called "mapped", into system area 126 of main memory 101. The information swapped into the system area may be data which is addressed by a program in the user area, or conversely may be an executable image of a file which is executed as described more completely below.

After the use of the information swapped into system area 126 is complete, a second interrupt 67h function 5 is sent to memory mapping executive 108, but this command indicates the pages are to be mapped from system area 126 to expanded memory. This operation was described above and is illustrated in Fig. 15.

After the pages are unswapped, the final command in the sequence, interrupt 67h, function 6 is sent to memory mapping executive 108. Memory mapping executive 108 in function 6, as illustrated in Fig. 17, first pushes the data segment onto the stack and sets the data segment register in a first save configuration 108-30. Validate handle 108-31 determines whether the handle passed in
interrupt 67h function 6 call was a valid handle. If the handle was valid, a second save configuration 108-32 stores the CPU registers on the stack and then transfers processing to handle search 108-33. Handle search 108-33 processes the handle page table to locate occurrences of the specified handle and reinitializes each occurrence of the handle to zero when it is found. Each time the handle is found, the page count is decremented by one, and the entry is cleared. After all of the handle entries have been located and the entries cleared, processing transfers to restore 108-34 which first decrements the handle counter and then pops the values for the registers on the stack back to the CPU, initializes the AH register to indicate a successful deallocation, and returns through return 108-35 to the processor.

The other functions supported by memory mapping executive 108, i.e., get status, get handle count, get handle pages, perform the function indicated by their title. Specifically, handle status simply returns a successful operation to interrupt 67h. The function get handle count complements the handle counter with 256 and subtracts one for handle 0 and returns. Finally, function 13, get handle pages, performs a function similar to deallocate handle except in this function after the handle is validated the page handle table is examined and the number of pages associated with the handle is only counted rather than deallocated as in function 6.

The memory mapping executive of this invention maps not only expanded memory as in prior art computers, but also the memory from either drive A, B, or C in the computer. Accordingly, the user has a new capability in managing memory allocations.

An executable image of a user program is loaded into system area as described above, but execution of the executable image in system area 126 requires use of a small program in user area 125. This program, for example, would include a command requesting creation of a
user process from the executable image. That is, in response to this command, the computer operating system, as described above, sends commands to information management executive 111 which in turn maps the executable image from one of the devices 102-105 into system area 126. The next command in this program initiates execution by pointing to the entry point of the executable image. Specifically, CS:IP is set so that CS points to the location of the executable image within the page frame after adjustment for any Header or PHB, as shown in Tables 2 and 3. IP is the main entry point offset for the executable image.

The embodiment of this invention, as described above, was included within a computer architecture functionally equivalent to an IBM XT computer architecture and accordingly, was written in assembly language for the Intel 8088 microprocessor. However, this embodiment was illustrative only of the principles of the invention and was not intended to limit the scope of the invention to the particular embodiment described. In view of this disclosure, those skilled in the art can implement the novel memory management executive of this invention in other computer architectures and in other computer programming languages.
I claim:

1. A system comprising;
   means for issuing I/O mapped commands; and
   information management means, operatively
   coupled to said issuing means, for converting said
   I/O mapped commands to memory mapped commands.

2. The system of Claim 1, said information
   management means further comprising:
   means for performing operations specified by
   said memory mapped command.

3. The system of Claim 2 further comprising:
   memory mapped peripheral storage means, wherein
   said information management means, in response to an
   I/O command, converts said I/O command to a memory
   mapped command and performs the operations
   corresponding to said memory mapped command using
   said memory mapped peripheral storage means.

4. The system of Claim 3, said information
   management means further comprising:
   means for swapping information in memory between
   said peripheral storage means and another memory.

5. The system of Claim 1 further comprising:
   memory means;
   memory mapped peripheral storage means
   operatively coupled to said memory means; and
   means for issuing memory mapping commands, each
   of said memory mapped commands specifying one of
   (i) information in said memory means that is to be
   transferred to said peripheral storage means and
   (ii) information in said peripheral storage means
that is to be transferred to said memory means.

6. The system of Claim 5, said information management means further comprising:
   memory mapping means, operatively coupled to said means for issuing memory mapping commands, for performing operations specified in a memory mapping command.

7. The system of Claim 6, said memory mapping means further comprising:
   means, operatively coupled to said memory means and to said memory mapped peripheral storage means, for swapping information between said memory means and said peripheral storage means.

8. A system comprising:
   memory means;
   memory mapped peripheral storage means;
   means, operatively coupled to said memory means and said peripheral storage means, for issuing I/O mapped commands wherein each I/O command specifies operations for an I/O mapped peripheral storage device;
   means, operatively coupled to said memory means and said peripheral storage means, for issuing memory mapping commands, each of said memory mapped commands specifying one of (i) information in said memory means that is to be transferred to said peripheral storage means and (ii) information in peripheral storage means to be transferred to said memory means; and
   information management means, operatively coupled to said I/O command issuing means and to said memory mapping command issuing means, for converting said I/O mapped commands to memory mapped commands and for performing memory mapped commands for said
memory mapped peripheral storage means.

9. The system of Claim 8, said information management means further comprising: means for swapping information between said memory means and said peripheral storage means.

10. The system of Claim 3, Claim 5, or Claim 8 wherein said memory mapped peripheral storage means comprises read-only memory means.

11. The system of Claim 10 wherein said read-only memory means includes a memory structure with a logical structure.

12. The system of Claim 11 wherein said logical structure comprises means for identifying the number of executable images stored in said memory structure.

13. The system of Claim 12, said logical structure further including means for identifying each executable image stored in said read-only memory.

14. The system of Claim 11 wherein said logical structure includes means for locating information stored in said read-only memory.

15. The system of Claim 14 wherein said means for locating information includes a boot sector.

16. The system of Claim 15 wherein said means for locating information further includes a file access table.

17. The system of Claim 3, Claim 5, or Claim 8 wherein said memory means includes a read-only memory structure with (i) a first logical structure including means for identifying the number of executable images
stored in said read-only memory means and (ii) a second logical structure for locating information including a boot sector.

18. The system of Claim 3, Claim 5, or Claim 8 wherein said memory mapped peripheral storage means comprises random access memory means.

19. The system of Claim 18 wherein said random access memory means includes a memory structure with a logical structure for locating information in said memory structure.

20. The system of Claim 19 wherein said means for locating information includes a boot sector.

21. The system of Claim 20 wherein means for locating information further includes a file access table.

22. The system of Claim 18 wherein said random access memory comprises at least a first random access memory means and a second random access memory means.

23. The system of Claim 22 wherein said first and second random access memory means both include a memory structure with a logical structure for locating information stored in said memory means.

24. The system of Claim 23 wherein said means for locating information includes a boot sector.

25. The system of Claim 24 wherein means for locating information further includes a file access table.

26. The system of Claim 3, Claim 5, or Claim 8 wherein said memory mapped peripheral storage means comprises random access memory means and read-only memory
27. In a computer system using I/O mapped commands to address peripheral storage units, a method for addressing memory mapped peripheral storage means comprising the steps of:
   converting each I/O mapped command to a memory mapped command; and
   performing said memory mapped command using said memory mapped peripheral storage means.

28. In a computer system using I/O mapped commands to address peripheral storage means, a method for enhancing the operational performance of said computer system and reducing the power consumption of said computer system comprising the steps of:
   operatively coupling at least one memory mapped peripheral storage means to said computer system;
   converting each I/O mapped command to a memory mapped command; and
   performing said memory mapped command using said memory mapped peripheral storage means.

29. The method of Claim 28 further comprising the step of:
   using said memory mapped peripheral storage means for memory mapping operations between said peripheral storage means and another memory in said computer system.

30. The method of Claim 28 wherein said peripheral storage means comprises read-only memory means.

31. The method of Claim 30 further comprising the step of configuring said read-only memory means with a logical structure.
32. The method of Claim 28 wherein said peripheral storage means comprises random access memory means.

33. The method of Claim 32 further comprising the step of configuring said random access memory means with a 5 logical structure.

34. The method of Claim 28 wherein said peripheral storage means comprises random access memory means and read-only memory means.

35. The method of Claim 34 further comprising the step of configuring each of said memory means with a logical structure.

36. The method of Claim 31 or Claim 35 wherein said logical structure includes means for identifying the number of executable images stored in said read-only memory.

37. The method of Claim 36 wherein said logical structure further includes means for identifying each executable image stored in said read-only memory.

38. The method of Claim 31, Claim 33, or Claim 35 wherein said logical structure includes means for locating information stored in said memory means.

39. The method of Claim 38 wherein said means for locating information includes a boot sector.

40. The method of Claim 39 wherein said means for locating information includes a file access table.

41. The method of Claim 31 or Claim 35 wherein read-only memory means logical structure includes a first logical structure including means for identifying the
number of executable images stored in said read-only memory means and a second logical structure for locating information including a boot sector.

42. A computer system comprising:

memory means;

memory mapped peripheral storage means operatively coupled to said memory means;

means for issuing memory mapping commands, each of said memory mapped commands specifying one of (i) information in said memory means that is to be transferred to said peripheral storage means and (ii) information in said peripheral storage means that is to be transferred to said memory means; and

memory mapping means, operatively coupled to said means for issuing memory mapping commands, to said memory means, and to said peripheral storage means, for performing operations specified in a memory mapping command.

43. The computer system of Claim 42, said memory mapping means further comprising:

means for swapping information between said memory means and said peripheral storage means.

44. The computer system of Claim 43 wherein said memory mapped peripheral storage means comprise a plurality of peripheral storage means and further wherein at least one of said peripheral storage means includes a nonvolatile random access memory.

45. The computer system of Claim 43 wherein said memory mapped peripheral storage means comprise a plurality of peripheral storage means and further wherein at least one of said peripheral storage means includes a read-only memory.
46. The computer system of Claim 43 wherein said memory mapped peripheral storage means comprise a plurality of peripheral storage means and further wherein said plurality of peripheral storage means includes a nonvolatile random access memory and at least one peripheral storage means having a read-only memory.

47. The computer system of Claim 43 having memory mapped peripheral storage means wherein said memory mapped peripheral storage means includes a memory structure with a logical structure.

48. The computer system of Claim 47 wherein said memory structure comprises a nonvolatile random access memory structure.

49. The computer system of Claim 47 wherein said memory structure comprises a read-only memory structure.

50. The computer system of Claim 48 or Claim 49 wherein said logical structure comprises means for identifying the number of executable images stored in said memory structure.

51. The computer system of Claim 50, said logical structure further including means for identifying each executable image stored in said read-only memory.

52. The computer system of Claim 48 or Claim 49 wherein said logical structure includes means for locating information stored in said memory structure.

53. The computer system of Claim 52 wherein said means for locating information includes a boot sector.

54. The computer system of Claim 53 wherein said means for locating information further includes a file
access table.

55. The computer system of Claim 49 wherein said logical structure comprises (i) a first logical structure including means for identifying the number of executable images stored in said read-only memory means and (ii) a second logical structure for locating information including a boot sector.

56. A peripheral information storage device for a computer system comprising a memory mapped peripheral storage means having a logical structure.

57. The device of Claim 56 wherein said memory mapped peripheral storage means comprises read-only memory means.

58. The device of Claim 56 wherein said memory mapped peripheral storage means comprises random access memory means.

59. The device of Claim 56 wherein said memory mapped peripheral storage means comprises random access memory means and read-only memory means.

60. The device of Claim 57 or Claim 59 wherein said logical structure includes means for identifying the number of executable images stored in said read-only memory.

61. The device of Claim 60 wherein said logical structure further includes means for identifying each executable image stored in said read-only memory.

62. The device of Claim 57, Claim 58, or Claim 59 wherein said logical structure includes means for locating information stored in said memory means.
63. The device of Claim 62 wherein said means for locating information includes a boot sector.

64. The device of Claim 62 wherein said means for locating information includes a file access table.

65. The device of Claim 57 or Claim 59 wherein said read-only memory means logical structure includes a first logical structure including means for identifying the number of executable images stored in said read-only memory means and a second logical structure for locating information including a boot sector.

66. A method for configuring a memory mapped peripheral storage device comprising:
   formatting said memory mapped peripheral storage device with a logical structure.

67. The method of Claim 66 wherein said step of formatting includes providing means for identifying the number of executable images stored in said memory map peripheral storage device.

68. The method of Claim 67 wherein said step of formatting further comprises providing means for identifying each executable image stored in said memory map peripheral storage device.

69. The method of Claim 66 wherein said formatting step further comprises providing means for locating information stored in said memory map peripheral storage device.

70. The method of Claim 69 wherein said step of providing means for locating information further comprises a boot sector.
71. The method of Claim 70 wherein said step of providing means for locating information further comprises providing a file access table.
FIGURE 2A
PRIOR ART
ADDRESS

A0000
Or Top of
User Area

10000 to 14000

00400

00000

MEMORY USE

Transient
Program
Area

COMMAND.COM

Device Drivers

DOS Buffers

DOS Kernel

BIOS Data Area

Int Vectors

25A

User
Area
25

FIGURE 2B
PRIOR ART
FIGURE 6

104D

104D - 2
104D - 3
104D - 4

BOOT FAT1 FAT2 DIRECTORY
FIGURE 7A

104E-1
104E-2
104E-3

104E