CP/M Plus

(CP/M (R) Version 3)
Operating System

Programmers Guide

COPYRIGHT

DISCLAIMER

NOTICE TO USER

From time to time changes are made in the filenames and in the files actually included on the distribution disk. This manual should not be construed as a representation or warranty that such files or facilities exist on the distribution disk or as part of the materials and programs distributed. Most distribution disks include a "README.DOC" file. This file explains variations from the manual which do constitute modification of the manual and the items included therewith. Be sure to read this file before using the software.

TRADEMARKS

The CPIM Plus (CP/M Version 3) Operating System Programmer's Guide was printed in the United States of America.

First Edition: January 1983
Second Edition: April 1983

Foreword

CP/M@ 3 is a microcom uter operating system designed for the Intelg 8080, Intel 8085, or other compatible microprocessor. To run CP/M 3, your computer must have an ASCII console, which includes a keyboard and screen, or another display device, from one to sixteen disk drives and a minimum of 32K of memory space. To support additional memory beyond the 64K addressing limit of the processors listed above, CP/M 3 can also support bank-switched memory. The minimum memory requirement for a banked system is 96K.

This manual describes the programming environment of CP/M 3, and is written for experienced programmers who are writing application software in the CP/M 3 environment. It assumes you are familiar with the system features and utilities described in the CP/M Plus (CP/M Version 3) Operating System User's Guide and the Programmer's Utilities Guide for the CP/M Family of Operating Systems. It also assumes that your CP/M 3 system has been customized for your computer's hardware and is executing as described in the CP/M Plus (CP/M Version 3) Operating System User's Guide. If you need to customize your system, please refer to the CP/M Plus (CP/M Version 3) Operating System System Guide.

Section 1 of this manual describes the components of the operating system, where they reside in memory, and how they work together to provide a standard operating environment for application programs. Section 2 describes how an application pro- gram can call on CP/M 3 to perform serial input and output and manage disk files. Section 3 provides a detailed description of each operating system function. Section 4 presents example programs.

The CP/M Plus (CP/M Version 3) Operating System Programmer's Guide contains five appendixes. Appendix A describes the CP/M 3 System Control Block, and defines its fields. Appendix B supplies the format for the Page Relocatable Program. Appen- dix C tells you how to generate System Page Relocatable files. Appendix D lists the ASCII Symbol Table, and Appendix E summarizes BDOS functions.

Table of Contents

Table of Contents (continued)

Appendixes

Table of Contents (continued)

Tables

Table of Contents (continued)

Figures

Section 1

Introduction to CP/M 3

This section introduces you to the general features of CP/M 3 with an emphasis on how CP/M 3 organizes your computer's memory. The section begins by describing the general memory organization of banked and nonbanked systems and defines the programming environment they have in common. It then shows how CP/M 3 defines memory space into standard regions for operating system modules and executing programs. Subsequent paragraphs describe the components of the operating system, how they communicate with each other and the application program, and in greater detail where each component and program is located in memory. After a brief introduction to disk organization, the final section gives examples of system operation.

CP/M 3 is available in two versions: a version that supports bank-switched memoy, and a version that runs on nonbanked systems. CP/M 3 uses the additional memory available in banked systems to provide functions that are not present in the nonbanked version. For example, the banked version of CP/M 3 supports file passwords; the nonbanked version does not. However, because a nonbanked system treats passwords the same way as a banked system does when password protection is not enabled, an application program can run under either system without modification.

1.1 Banked and Nonbanked Memory Organization

The memory organization for a nonbanked CP/M 3 system is very simple, as shown in Figure 1-1.

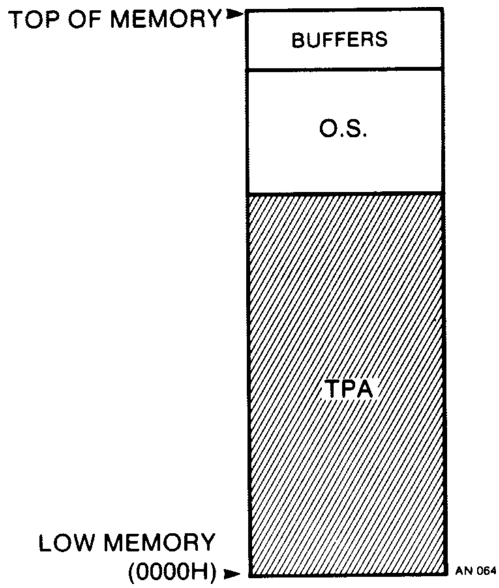

Figure 1-1. Nonbanked System Memory Organization

In the nonbanked organization, physical memory consists of a single, contiguous region addressable from OOOOH up to a maximum of OFFFFH (64K-1). The shaded region below the operating system represents the memory space available for the loading and execution of transient programs. The clear area above the operating system represents space that GENCPM can allocate to the operating system for disk record buffers and directory hash tables, as described in the CP/M Plus (CP/M Version 3) Operating System System Guide. The minimum size of this area is determined by the specific hardware requirements of the host microcomputer system.

To expand memory capacity beyond the 64K address space of an 8-bit microprocessor, CP/M 3 supports bank-switched memory in a special version called the banked system. In the banked version, the operating system is divided into two modules: the resident portion and the banked portion. The resident portion resides in common memory; the banked portion resides just below the top of banked memory in Bank 0. Figure 1-2 shows memory organization under the banked system.

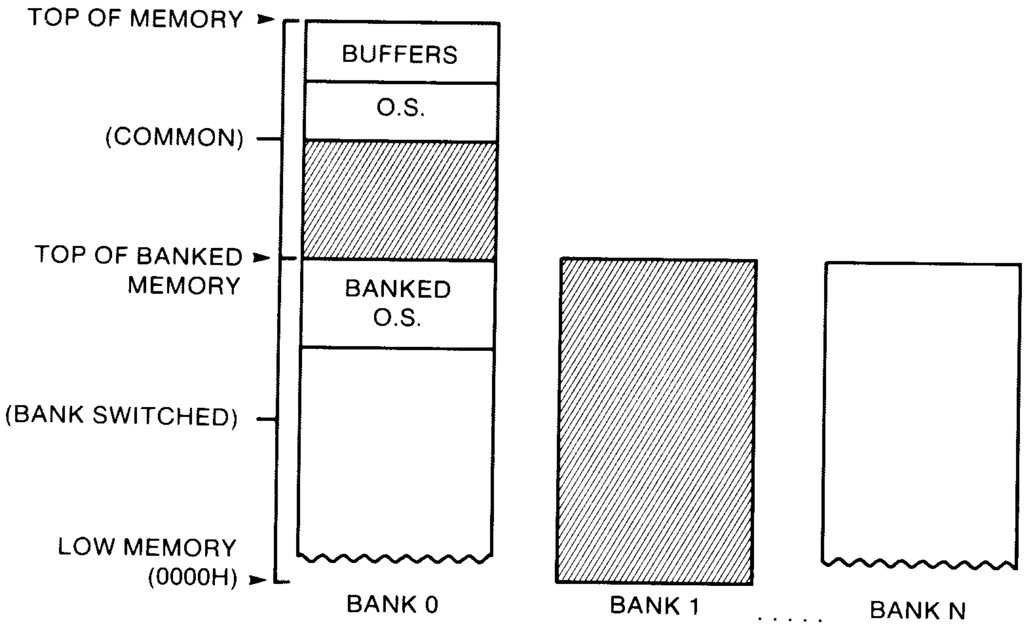

Figure 1-2. Banked System Memory Organization

In Figure 1-2, Bank 0 is switched in or in context. The top region of memory, the common region, is always in context; that is, it can always be referenced, no matter what bank is switched in. Figure 1-3 shows memory organization when Bank 1 is in context.

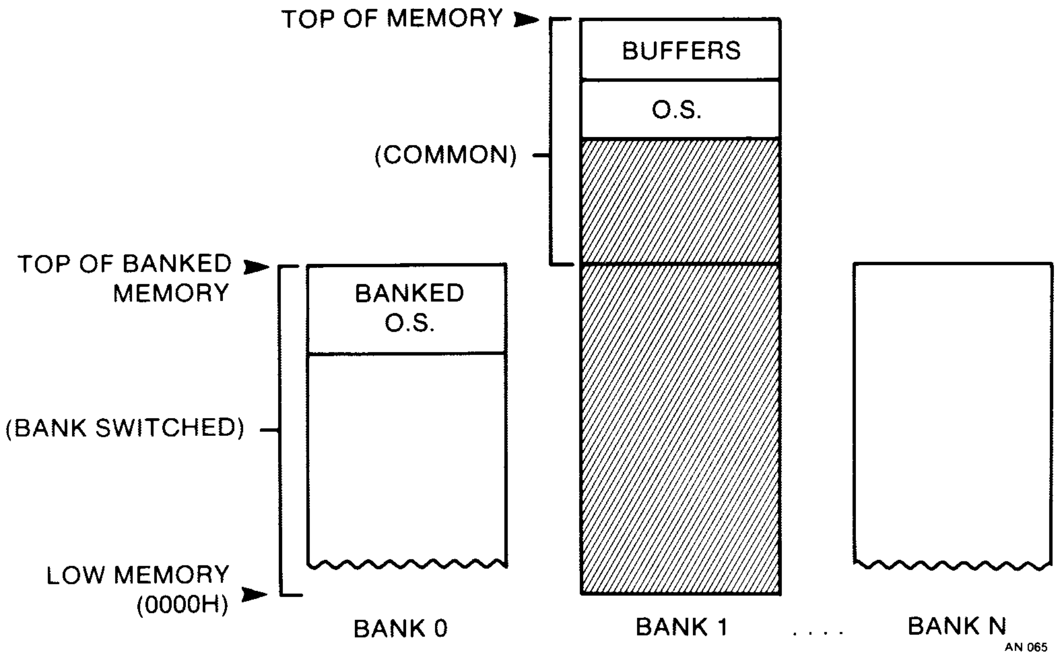

Figure 1-3. Banked Memory with Bank 1 in Context

From a transient programs perspective, Bank 1 is always in context. The operating system can switch to Bank 0 or other banks when performing operating system functions without affecting.the execution of the transient program. Any bank-switching performed by the operating system is completely transparent to the calling program. Because the major portion of the operating system resides in Bank 0 in banked systems, more memory space is available for transient programs in banked CP/M 3 systems than in nonbanked systems.

The operating system uses the clear areas in Figures 1-2 and 1-3 for disk record buffers and directory hash tables. The clear area in the common region above the operating system represents space that can be allocated for data buffers by GENCPM. Again, the minimum size of this area is determined by the specific hardware requirements of the host microcomputer system.

The banked version of CP/M 3 requires a minimum of two banks, Bank 0 and Bank 1, and can support up to 16 banks of memory. Bank numbers are generally arbitrary with the following exceptions: Bank 0 is the system bank and is in context when CP/M 3 is started. Bank 1 is the transient program bank, and must be contiguous from location zero to the top of banked memory. This requirement does not apply to the other banks. However, common memory must be contiguous.

The size of the common region is typically 16K. The only size requirement on the common region is that it must be large enough to contain the resident portion of the operating system. The maximum top of memory address for both banked and nonbanked systems is 64K-1 (OFFFFH).

In summary, no matter how physical memory is configured, or whether the operating system is banked or nonbanked, CP/M 3 always organizes memory logically so that to a transient program in any CP/M 3 system, memory appears as shown in Figure 1-4.

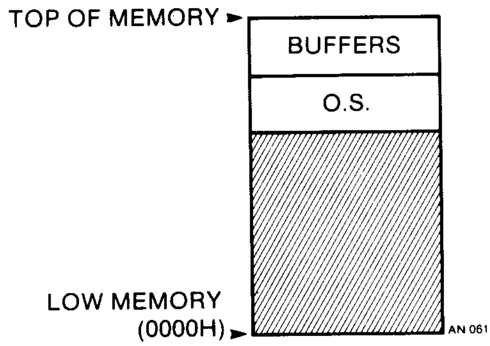

Figure 1-4. CP/M 3 Logical Memory Organization

1.2 System Components

Functionally, the CP/M 3 operating system is composed of distinct modules. Transient programs can communicate with these modules to request system services. Figure 1-5 shows the regions where these modules reside in logical memory. Note that from the transient program's perspective, Figure 1-5 is just a more detailed version of Figure 1-4.

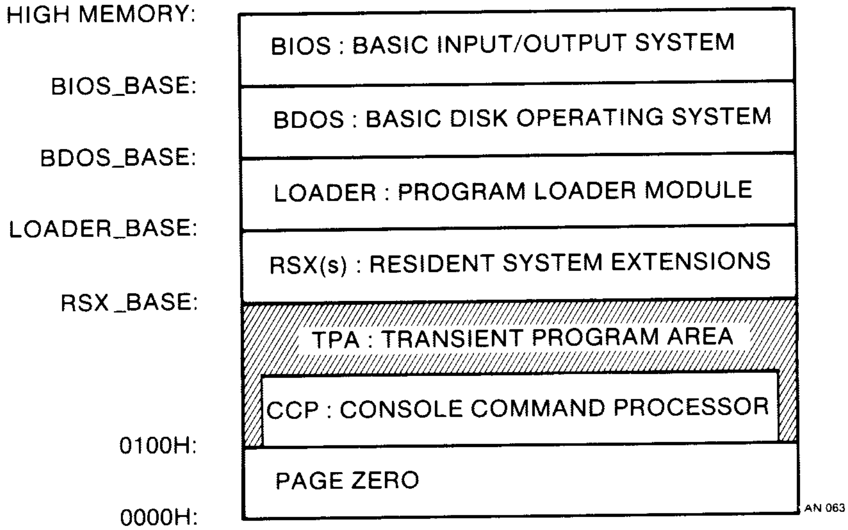

Figure 1-5. System Components and Regions in Logical Memory

The Basic Input/Output System, BIOS, is a hardware-dependent module that defines the low-level interface to a particular computer system. It contains the device-driving routines necessary for peripheral device I/O.

The Basic Disk Operating System, BDOS, is the hardware-independent module that is the logical nucleus of CP/M 3. It provides a standard operating environment for transient programs by making services available through numbered system function calls.

The LOADER module handles program loading for the Console Command Processor and transient programs. Usually, this module is not resident when transient programs execute. However, when it is resident, transient programs can access this module by making BDOS Function 59 calls.

Resident System Extensions, RSXS, are temporary additional operating system modules that can selectively extend or modify normal operating system functions. The LOADER module is always resident when RSXs are active.

The Transient Program Area, TPA, is the region of memory where transient programs execute. The CCP also executes in this region.

The Console Command Processor, CCP, is not an operating system module, but is a system program that presents a human-oriented interface to CP/M 3 for the user.

The Page Zero region is not an operating system module either, but functions primarily as an interface to the BDOS module from the CCP and transient programs. It also contains critical system parameters.

1.3 System Component Interaction and Communication

This section describes interaction and communication between the modules and regions defined in Section 1.2. The most significant channels of communication are between the BDOS and the BIOS, transient programs and the BDOS, and transient programs and RSXS.

The division of responsibility between the different modules and the way they communicate with one another provide three important benefits. First, because the operating system is divided into two modules-one that is configured for different ardware environments, and one that remains constant on every computer-CP/M 3 software is hardware independent; you can port your programs unchanged to different hardware configurations. Second, because all communication between transient programs and the BDOS is channeled through Page Zero, CP/M 3 transient programs execute, if sufficient memory is available, independent of configured memory size. Third, the CP/M 3 RSX facility can customize the services of CP/M 3 on a selective basis.

1.3.1 The BDOS and BIOS

CP/M 3 achieves hardware independence through the interface between the BDOS and the BIOS modules of the operating system. This interface consists of a series of entry points in the BIOS that the BDOS calls to perform hardware-dependent primitive functions such as peripheral device VO. For example, the BDOS calls the CONIN entry point of the BIOS to read the next console input character.

A system implementor can customize the BIOS to match a specific hardware environment. However, even when the BIOS primitives are customized to match the host computer's hardware environment, the BIOS entry points and the BDOS remain constant. Therefore, the BDOS and the BIOS modules work together to give the CCP and other transient programs hardware-independent access to CP/M 3's facilities.

1.3.2 Applications and the BDOS

Transient programs and the CCP access CP/M 3 facilities by making BDOS function calls. BDOS functions can create, delete, open, and close disk files, read or write to opened files, retrieve input from the console, send output to the console or list device, and perform a wide range of other services described in Section 3,"BDOS Functions."

To make a BDOS function call, a transient program loads CPU registers with specific entry parameters and calls location 0005H in Page Zero. If RSXs are not active in memory, location 0005H contains a jump instruction to location BDOS_base + 6. If RSXs are active, location 0005H contains a jump instruction to an address below BDOS -base. Thus, the Page Zero interface allows programs to run without regard to where the operating system modules are located in memory. In addition, transient programs can use the address at location 0006H as a memory ceiling.

Some BDOS functions are similar to BIOS entry points, particularly in the case of simple device I/O. For example, when a transient program makes a console output BDOS function call, the BDOS makes a BIOS console output call. In the case of disk I/O, however, this relationship is more complex. The BDOS might call many BIOS entry points to perform a single BDOS file I/O function.

Transient programs can terminate execution by jumping to location OOOOH in the Page Zero region. This location contains a jump instruction to BIOS base+3, which contains a jump instruction to the BIOS warm start routine. The BIOS warm start routine loads the CCP into memory at location 100H and then passes control to it.

The Console Command Processor is a special system program that executes in the TPA and makes BDOS calls 'ust like an application program. However, the CCP has a unique role: it gives the user access to operating system facilities while transient programs are not executing. It includes several built-in commands, such as TYPE and DIR, that can be executed directly without having to be loaded from disk. When the CCP receives control, it reads the uscr's command lines, distinguishes between built-in and transient commands, and when necessary, calls upon the LOADER module to load transient programs from disk into the TPA for execution. Section 1.6.2 describes CCP operation in detail.

1.3.3 Applications and RSXs

A Resident System Extension is a temporary additional operating system module. An RSX can extend or modify one or more operating system functions selectively. As with a standard BDOS function, a transient program accesses an RSX function through a numbered function call.

At any one time there might be zero, one, or multiple RSXs active in memory. When a transient program makes a BDOS function call, and RSXs are active, each RSX examines the function number of the call. If the function number matches the function the RSX is designed to extend or modify, the RSX performs the requested function. Otherwise, the RSX passes the function request to the next RSX. Nonintercepted functions are eventually passed to the BDOS for standard execution.

RSXs are loaded into memory when programs containing RSXs are loaded. The CP/M 3 utility, GENCOM, can attach RSXs to program files. When attaching RSXS, GENCOM places a special one page header at the beginning of the program file. The CCP reads this header, learns that a program has attached RSXS, and loads the RSXs accordingly. The header itself is not loaded into memory; it merely indicates to the CCP that RSX loading is required.

The LOADER module is a special type of RSX that supports BDOS function 59, Load Overlay. It is always resident when RSXs are active. To indicate RSX support is required, a program that calls function 59 must have an RSX header attached by GENCOM, even if the program does not require other RSXS. When the CCP encounters this type of header in a program file when no RSXs are active, it sets the address at location 0006H in Page Zero to LOADER-base + 6 instead of BDOS_base + 6.

1.4 Memory Region Boundaries

This section reviews memory regions under CP/M 3, and then describes some details of region boundaries. It then relates the sizes of various modules to the space avialable for the execution of transient programs. Figure 1-6 reviews the location of regions in logical memory.

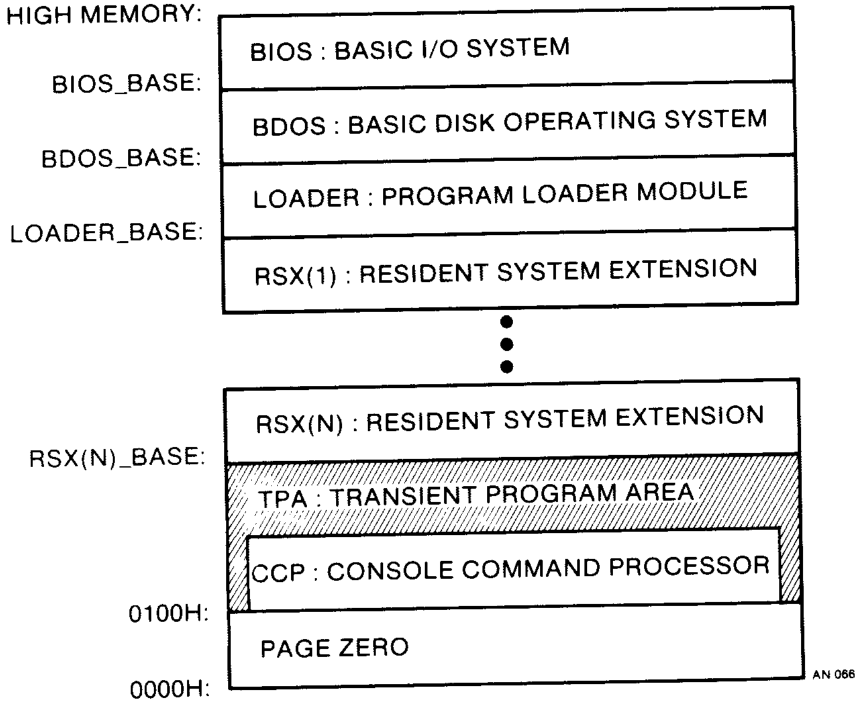

Figure 1-6. System Modules and Regions in Logical Memory

First note that all memory regions in CP/M 3 are page-aligned. This means that regions and operating system modules must begin on a page boundary. A page is defined as 256 bytes, so a page boundary always begins at an address where the low-order byte is zero.

The term High Memory in Figure 1-6 denotes the high address of a CP/M 3 system. This address may fall below the actual top of memory address if space above the operating system has been allocated for directory hashing or data buffering by GENCPM. The maximum top of memory address for both banked and nonbanked systems is 64K-1 (OFFFFH).

T'he labels BIOS-base, BDOS_base, and LOADER_base represent the base addresses of the operating system regions. These addresses always fall on page boundaries. The size of the BIOS region is not fixed, but is determined by the requirements of the host computer system.

The size of the BDOS region differs for the banked and nonbanked versions of CP/M 3. In the banked version, the resident BDOS size is 6 pages, 1.5K. In the nonbanked system, the BDOS size ranges from 31 pages, 7.75K, to 33 pages, 8.25K, depending on system generation options and BIOS requirements.

RSXs are page aligned modules that are stacked in memory below LOADER_base in memory. In the configuration shown in Figure 1-6, location 0005H of Page Zero contains a jump to location RSX(N)_base + 6. Thus, the memory ceiling of the TPA region is reduced when RSXs are active.

Under CP/M 3. the CCP is a transient program that the BIOS loads into the TPA region of memory at system cold and warm start. The BIOS also loads the LOADER module at this time, because the LOADER module is attached to the CCP. When the CCP gains control, it relocates the LOADER module just below BDOS_base. The LOADER module handles program loading for the CCP. It is three pages long.

The maximum size of a transient program that can be loaded into the TPA Is limited by LOADER-base because the LOADER cannot load a program over itself. Transient programs may extend beyond this point, however, by using memory above LOADER - base for uninitialized data areas such as I/O buffers. Programs that use memory above BDOS-base cannot make BDOS function calls.

1.5 Disk and Drive Organization and Requirements

CP/M 3 can support up to sixteen logical drives, identified by the letters A through PI, with up to 512 megabytes of storage each. A logical drive usually corresponds to a physical drive on the system, particularly for physical drives that support removable media such as floppy disks. High-capacity hard disks, however, are commonly divided up into multiple logical drives. Figure 1-7 illustrates the standard organization of a CP/M 3 disk.

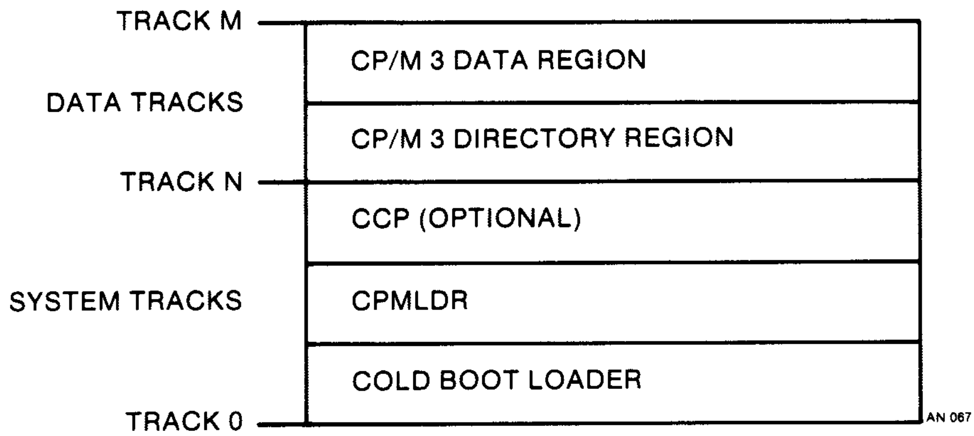

Figure 1-7. Disk Organization

In Figure 1-7, the first N tracks are the system tracks. System tracks are required only on the disk used by CP/M 3 during system cold start or warm start. The contents of this region are described in Section 1.6.1. All normal CP/M 3 disk access is directed to the data tracks which CP/M 3 uses for file storage.

The data tracks are divided into two regions: a directory area and a data area. The directory area defines the files that exist on the drive and identifies the data space that belongs to each file. The data area contains the file data defined by the directory. If the drive has adequate storage, a CP/M 3 file can be as large as 32 megabytes.

The directory area is subdivided into sixteen logically independent directories. These directories are identified by user numbers 0 through 15. During system operation, CP/M 3 runs with the user number set to a single value. The user number can be changed at the console with the USER command. A transient program can change the user number by calling a BDOS function.

The user number specifies the currently active directories for all the drives on the svstem. For example, a PIP command to copy a file from one disk to another gives t@e destination file the same user number as the source file unless the PIP command is modified by the [G] option.

The directory identifies each file with an eight-character filename and a three-character filetype. Together, these fields must be unique for each file. Files with the same filename and filetype can reside in different user directories on the same drive without conflict. Under the banked version of CP/M 3, a file can be assigned an eight-character password to protect the file from unauthorized access.

All BDOS functions that involve file operations specify the requested file by filename and filetype. Multiple files can be specified by a technique called ambiguous reference, which uses question marks and asterisks as wildcard characters to give CP/M 3 a pattern to match as it searches the directory. A question mark in an ambiguous reference matches any value in the same position in the directory filename or filetype field. An asterisk fills the remainder of the filename or filetype field of the ambiguous reference with question marks. Thus, a filename and filetype field of all question marks, ????????.???, equals an ambiguous reference of two asterisks, and matches all files in the directory that belong to the current user number.

The CP/M 3 file system automatically allocates directory space and data area space when a file is created or extended, and returns previously allocated space to free space when a file is deleted or truncated. If no directory or data space is available for a requested operation, the BDOS returns an error to the calling program. In general, the allocation and deallocation of disk space is transparent to the calling program. As a result, you need not be concerned with directory and drive organization when using the file system facilities of CP/M 3.

1.6 System Operation

This section introduces the general operation of CP/M 3. This overview covers opics concerning the CP/M 3 system components, how they function and how they interact when CP/M 3 is running. This section does not describe the total functionality of CP/M 3, but simply introduces basic CP/M 3 operations.

For the purpose of this overview, CP/M 3 system operation is divided into five categories. First is system cold start, the process that begins execution of the operating system. This procedure ends when the Console Command Processor, CCP, is loaded into memory and the system prompt is displayed on the screen. Second is the operation of the CCP, which provides the user interface to CP/M 3. Third is transient program initiation, execution and termination. Fourth is the way Resident System Ex,ensions run under CP/M 3. The fifth and final category describes the operation of the CP/M 3 SUBMIT utility.

1.6.1 Cold Start Operation

The cold start procedure is typically executed immediately after the computer is turned on. The cold start brings CP/M 3 into memory and gives it control of the computer's resources. Cold start is a four-stage procedure.

In the first stage, a hardware feature, or ROM-based software associated with system reset, loads a small program, called the Cold Boot Loader, into memory from the system tracks of drive A (see figure 1-6). The Cold Boot Loader is usually 128 or 256 bytes long.

The Cold Boot Loader performs the second stage of the cold start process. It loads the CP/M 3 loader program, CPMLDR, into memory from the system tracks of the system disk and passes control to it. During this stage, the Cold Boot Loader can also perform other tasks, such as initializing hardware dependent I/O ports.

CPMLDR performs the third stage in the cold start process. First, it reads the CPM3.SYS file from the data area of the disk. The CPM3.SYS file, which is created by the CP/M 3 system generation utility GENCPM, contains the BDOS and BIOS system components and information indicating where these modules are to reside in memory. Once CPMLDR has loaded the BDOS and BIOS into memory, it sends a sign-on message to the console and passes control to the BIOS Cold Boot entry point. If specified as a GENCPM option, CPMLDR can also display a memory map of the CP/M 3 system.

CPMLDR is a small, self-contained version of CP/M 3 that supports only console output and sequential file input. Consistent with CP/M 3's organization, it contains two modules, an invariant CPMLDR-BDOS, and a variant CPMLDR -BIOS that is adapted to match the host microcomputer hardware environment. Cold start initialization of VO ports and similar functions can also be performed in the CPMLDR-BIOS module during the third stage of cold start.

In the banked version of CP/M 3, these first three stages of the cold boot procedure are performed with Bank 0 in context. The BIOS Cold Start function switches in Bank 1 before proceeding to stage four.

The fourth and final stage in the cold start procedure is performed by the BIOS Cold Start function, Function 0. The entry point to this function is located at BIOS -base as described in Section 1.4. The BIOS Cold Start function begins by performing any remaining hardware initialization, and initializing Page Zero. To initialize Page Zero, the BIOS Cold Start function places a jump to BIOS base + 3, the BIOS Warm Start entry point, at location OOOOH, and a jump to BDOS-base + 6, the BDOS entry point, at location 0005H in memory.

The BIOS Cold Start function completes the fourth stage by loading the CCP into the TPA region of memory and passing control to it. The CCP can be loaded from one of two locations. If there is sufficient space in the system tracks for the CCP, it is usually loaded from there. If there is not enough space in the system tracks, the BIOS Cold Start function can read the CCP from the file CCP.COM.

On some banked systems, the CCP is also copied to an alternate bank, so that warm start operations can copy the CCP into the TPA from memory. This speeds up the system warm start operation, and makes it possible to warm start the system without having to access a system disk.

When the CCP gains control, it displays a prompt that references the default disk. If a PROFILE.SUB submit file is present on the default drive, the CCP executes this submit file before prompting the user for a command.

At this point, the cold start procedure is complete. Note that the user number is set to zero when CP/M 3 is cold started. However, the PROFILE submit file can set the user number to another value if this is desirable.

The cold start procedure is designed so that the system tracks need to be initialized only once. This is accomplished because the system track routines are independent of the configured memory size of the CP/M 3 system. The Cold Boot Loader loads CPMLDR into a constant location in memory. This location is chosen when the system is configured. However, CPMLDR locates the BDOS and BIOS system components in memory as specified by the CPM3.SYS file. The CCP always executes at location 100H in the TPA. Thus, CP/M 3 allows the user to generate a new system with GENCPM, and then run it without having to update the system tracks of the system disk.

1.6.2 CCP Operation

The Console Command Processor provides the user access to CP/M 3 facilities when transient programs are not running. It also reads the user's command lines, differentiates between built-in commands and transient commands, and executes the commands accordingly.

This section describes the responsibilities and capabilities of the CCP in some detail. The section begins with a description of the CCP's activities when it first receives control from the Cold Start procedure. The section continues with a general discussion of built-in commands, and concludes with a step-by-step description of the procedure the CCP follows to execute the user's commands.

When the CCP gains control following a cold start procedure, it displays the system prompt at the console. This signifies that the CCP is ready to execute a command. The system prompt displays the letter of the drive designated as the initial default drive during GENCPM operation. For example, if drive A was specified as the initial default drive, the CCP displays the following prompt:

```
A >
```

After displaying the system prompt, the CCP scans the directory of the default drive for the file PROFILE.SUB. If the file exists, the CCP creates the command line SUBMIT PROFILE; otherwise the CCP reads the user's first command line by making a BDOS Read Console Buffer function call (BDOS Function 10).

The CCP accepts two different command forms. The simplest CCP command form changes the default drive. The following example illustrates a user changing the default drive from A to B.

```
A>B:
B >
```

This command is one of the CCP's built-in commands. Built-in commands are part of the CCP. They reside in memory while the CCP is active, and therefore can be executed without referencing a disk.

The second command form the CCP accepts is the standard CP/M command line. A standard CP/M command line consists of a command keyword followed by an optional command tail. The command keyword and the command tall can be typed in any combination of upper-case and lower-case letters; the CCP converts all letters n the command line to upper-case. The following syntax defines the standard CP/M command line:

<command> <command tail>

where

<command>	=> <filespec> or <built-in>
<command tail>	=> (no command tail) or <filespec> or <filespec><delimiter><filespec>
<filespec>	=> {d:}filename{.typ}{;password}
<built-in>	=> one of the CCP built-in commands
<delimiter>	=> one or more blanks or a tab or one of the following:"=,[]<> "
d:	=> CP/M 3 drive specification,"A" through"P"
filename	=> 1 to 8 character filename
typ	=> 1 to 3 character filetype
password	=> 1 to 8 character password value

Fields enclosed in curly brackets are optional. If there is no drive {d:} present in a file specification <filespec>, the default drive is assumed. If the type field {.typ} is omitted, a type field of all blanks is implied. Omitting the password field @;passwordl implies a password of all blanks. When a command line is entered at the console, it is terminated by a return or line-feed keystroke.

Transient programs that run under CP/M 3 are not restricted to the above command tail definition. However, the CCP only parses command tails in this format for transient programs. Transient programs that define their command tails differently must perform their own command tall parsing.

The command field must identify either a built-in command, a transient program, or a submit file. For example, USER is the keyword that identifies the built-in command that changes the current user number. The CP/M 3 CCP displays the user number in the system prompt when the user number is non-zero. The following example illustrates changing the user number from zero to 15.

```
B>USER 15
15B>
```

The following table summarizes the built-in commands.

Table 1-1. CP/M 3 Built-in Commands

Command	Meaning
DIR	displays a list of all filenames from a disk directory except those marked with the SYS attribute.
DIRSYS	displays a filename list of those files marked with the SYS attribute in the directory.
ERASE	erases a filename from a disk directory and releases the storage occupied by the file.
RENAME	renames a file.
TYPE	displays the contents of an ASCII character file at your console output device.
USER	changes from one user number to another.

Some built-in commands have associated command files which expand upon the options provided by the built-in command. If the CCP reads a command line and discovers the built-in command does not support the options requested in the command line,, the CCP loads the built-in function's corresponding command file to perform the command. The DIR command is an example of this type of command. Simple DIR commands are supported by the DIR built-in directly. More complex requests are handled by the DIR.COM utility.

All command keywords that do not identify built-in commands identify either a transient program file or a submit file. If the CCP identifies a command keyword as a transient program, the transient program file is loaded into the TPA from disk and executed. If it recognizes a submit file, the CCP reconstructs the command line into the following form:

SUBMIT <command> <command tail>

and attempts to load and execute the SUBMIT utility. Thus, the original command fie'ld becomes the first command tail field of the SUBMIT command. Section 1.6.5 describes the execution of CP/M 3's SUBMIT utility. The procedure the CCP follows o parse a standard command line and execute built-in and transient commands is described as follows:

1. The CCP parses the command line to pick up the command field.

2. If the command field is not preceded by a drive specification, or followed by a filetype or password field, the CCP checks to see if the command is a CCP built-in function. If the command is a built-in command, and the CCP can support the options specified in the command tail, the CCP executes the command. Otherwise, the CCP goes on to step 3.

3. At this point the CCP assumes the command field references a command file or submit file on disk. If the optional filetype field is omitted from the command, the CCP usually assumes the command field references a file of type COM. For example, if the command field is PIP, the CCP attempts to open the file PIP.COM.

   Optionally, the CP/M 3 utility SETDEF can specify that a filetype of SUB also be considered when the command filetype field is omitted. When this automatic submit option is in effect, the CCP attempts to open the command with a filetype of COM. If the COM file cannot be found, the CCP repeats the open op eration with a filetype of SUB. As an alternative, the order of open operations can be reversed so that the CCP attempts to open with a filetype of SUB first. In either case, the file that is found on disk first determines the filetype field that is ultimately associated with the command.
   If the filetype field is present in the command, it must equal COM, SUB or PRL. A PRL file is a Page Relocatable file used in Digital Research's multiuser operating system, MP/M. Under CP/M 3, the CCP handles PRL files exactly like COM files.

If the command field is preceded by a drive specification id:), the CCP attempts to open the command or submit file on the specified drive. Otherwise, the CCP attempts to open the file on the drives specified in the drive chain. The drive chain specifies up to four drives that are to be referenced in sequence for CCP open operations of command and submit files. If an open operation is unsuccessful on a drive in the drive chain because the file cannot be found, the CCP repeats the open operation on the next drive in the chain. This sequence of open operations is repeated until the file is found, or the drive chain is exhausted. The drive chain contains the current default drive as its only drive unless the user modifies the drive chain with the CP/M 3 SETDEF utility.

When the current user number is non-zero, all open requests that fail because the file cannot be found, attempt to locate the command file under user zero. If the file exists under user zero with the system attribute set, the file is opened from user zero. This search for a file under user zero is made by the BDOS Open File function. Thus, the user zero open attempt is made before advancing to the next drive in the search chain.
When automatic submit is in effect, the CCP attempts to open with the first filetype, SUB or COM, on all drives in the search chain before trying the second filetype.

In the banked system, if a password specified in the command field does not match the password of a file on a disk protected in Read mode, the CCP file open operation is terminated with a password error.
If the CCP does not find the command or submit file, it echoes the command line followed by a question mark to the console. If it finds a command file with a filetype of COM or PRL, the CCP proceeds to step 4. If it finds a submit file, it reconstructs the command line as described above, and repeats step 3 for the command, SUBMIT.COM.

4. When the CCP successfully opens the command file, it initializes the following Page Zero fields for access by the loaded transient program:

   0050H Drive that the command file was loaded from
   0051H Password address of first file in command tail
   0053H Password length of first file in command tail
   0054H Password address of second file in command tail
   0056H Password length of second file in command tail
   005CH Parsed FCB for first file in command tail
   006CH Parsed FCB for second file in command tail
   0080H Command tail preceded by command tail length

   Page Zero initialization is covered in more detail in Section 2.4.

5. At this point, the CCP calls the LOADER module to load the command file into the TPA. The LOADER module terminates the load operation if a read error occurs, or if the available TPA space is not large enough to contain the file. If no RSXs are resident in memory, the available TPA space is determined by the address LOADER - base because the LOADER cannot load over itself. Otherwise, the maximum TPA address is determined by the base address of the lowest RSX in memory.

6. Once the program is loaded, the LOADER module checks for a RSX header on the program. Programs with RSX headers are identified by a return instruction at location 100H.

   If an RSX header is present, the LOADER relocates all RSXs attached to the end of the program, to the top of the TPA region of memory under the LOADER module, or any other RSXs that are already resident. It also updates the address in location 0006H of Page Zero to address the lowest RSX in memory. Finally, the LOADER discards the RSX header and relocates the program file down one page in memory so that the first executable instruction resides at 100H.

7. After initializing Page Zero, the LOADER module sets up a 32-byte stack with the return address set to location OOOOH of Page Zero and jumps to location 100H. At this point, the loaded transient program begins execution.

When a transient program terminates execution, the BIOS warm start routine reloads the CCP into memory. When the CCP receives control, it tests to see if RSXs are resident in memory. If not, it relocates the LOADER module below the BDOS module at the top of the TPA region of memory. Otherwise, it skips this step because the LOADER module is already resident. The CCP execution cycle then repeats.

Unlike earlier versions of CP/M, the CCP does not reset the disk system at warm start. However, the CCP does reset the disk system if a CTRL-C is typed at the prompt.

1.6.3 Transient Program Operation

A transient program is one that the CCP loads into the TPA region of memory and executes. As the name transient implies, transient programs are not system resident. The CCP must load a transient program into memory every time the program is to be executed. For example, the utilities PIP and RMAC" that are shipped with CP/M 3 execute as transient programs; programs such as word processing and accounting packages distributed by applications vendors also execute as transient programs under CP/M 3.

Section 1.6.2 describes how the CCP prepared the CP/M 3 environment for the execution of a transient program. To summarize, the CCP initializes Page Zero to contain parsed command-line fields and sets up a 32-byte stack before jumping to location 0100H to pass control to the transient program. In addition, the CCP might also load RSXs attached to the command file into memory for access by the transient program.

Generally, an executing transient program communicates with the operating system only through BDOS function calls. Transient programs make BDOS function calls by loading the CPU registers with the appropriate entry parameters and calling location 0005H in Page Zero.

Transient programs can use BDOS Function 50, Call BIOS, to access BIOS entry points. This is the preferred method for accessing the BIOS; however, for compatibility with earlier releases of CP/M, transient programs can also make direct BIOS calls for console and list I/O by using the jump instruction at location OOOOH in Page Zero. But, to simplify portability, use direct BIOS calls only where the primitive level of functionality provided by the BIOS functions is absolutely required. For example, a disk formatting program must bypass CP/M's disk organization to do its job, and therefore is justified in making direct BIOS calls. Note however, that disk formatting programs are rarely portable.

A transient program can terminate execution in one of three ways: by jumping to location 0000H, by making a BDOS System Reset call, or by making a BDOS Chain To Program call. The first two methods are equivalent; they pass control to the BIOS warm start entry point, which then loads the CCP into the TPA, and the CCP prompts for the next command.

The Chain to Program call allows a transient program to specify the next command to be executed before it terminates its own execution. A Program Chain call executes a standard warm boot sequence, but passes the command specified by the terminating program to the CCP in such a way that the CCP executes the specified command instead of prompting the console for the next command.

Transient programs can also set a Program Return Code before terminating by making a BDOS Function 108 call, Get/Set Program Return Code. The CCP initializes the Program Return Code to zero, successful, when it loads a transient program, unless the program is loaded as the result of a program chain. Therefore, a transient program that terminates successfully can use the Program Return Code to pass a value to a chained program. If the program terminates as the result of a BDOS fatal error, or a CTRL-C entered at the console, the BDOS sets the return code to an unsuccessful value. All other types of program termination leave the return code at its current value.

The CCP has a conditional command facility that uses the Program Return Code. If a command line submitted to the CCP by the SUBMIT utility begins with a colon, the CCP skips execution of the command if the previous command set an unsuccessful Program Return Code. In the following example, the SUBMIT utility sends a command sequence to the CCP:

```
A>SUBMIT SUBFILE
A>COMPUTE RESULTS.DAT
A >: REPORT RESULTS. DA T
```

The CCP does not execute the REPORT command if the COMPUTE command ses an unsuccessful Program Return Code.

1.6.4 Resident System Extension Operation

This section gives a general overview of RSX use, then describes how RSXs are loaded, defines the RSX file structure, and tells how the LOADER module uses the RSX prefix and flags to manage RSX activity.

A Resident System Extension (RSX) is a special type of program that can be attached to the operating system to modify or extend the functionality of the BDOS. RSX modules intercept BDOS functions and either perform them, translate them into other BDOS functions, or pass them through untouched. The BDOS executes non-intercepted functions in the standard manner.

A transient program can also use BDOS Function 60, Call Resident System Extension, to call an RSX for special functions. Function 60 is a general purpose function that allows customized interfaces between programs and RSXS.

Two examples of RSX applications are the GET utility and the LOADER module. The GET.COM command file has an attached RSX, GET.RSX, which intercepts all console input calls and returns characters from the file specified in the GET command line. The LOADER module is another example of an RSX, but it is special because it supports Function 59, Load Overlay. It is always resident in memory when other RSXs are active.

RSXs are loaded into memory at program load time. As described in Section 1.6.2, after the CCP locates a command file, it calls the LOADER module to load the program into the TPA. The LOADER loads the transient program into memory along with any attached RSXS. Subsequently, the loader relocates each attached RSX to the top of the TPA and adjusts the TPA size by changing the jump at location 0005H in Page Zero to point to the RSX. When RSX modules reside in memory, the LOADER module resides directly below the BDOS, and the RSX modules stack downward from it.

The order in which the RSX modules are stacked affects the order in which they intercept BDOS calls. A more recently stacked RSX has precedence over an older RSX. Thus,, if two RSXs in memory intercept the same BDOS function, the more recently loaded RSX handles the function.

The CP/M 3 utility GENCOM attaches RSX modules to program files. Program files with attached RSXs have a special one page header that the LOADER recognizes when it loads the command file. GENCOM can also attach one or more RSXs to a null command file so that the CCP can load RSXs without having to execute a transient program. In this case, the command file consists of the RSX header followed by the RSXS.

RSX modules are Page Relocatable, PRL, files with the file type RSX. RSX files must be page relocatable because their execution address is determined dynamically by the LOADER module at load time. RSX files have the following format:

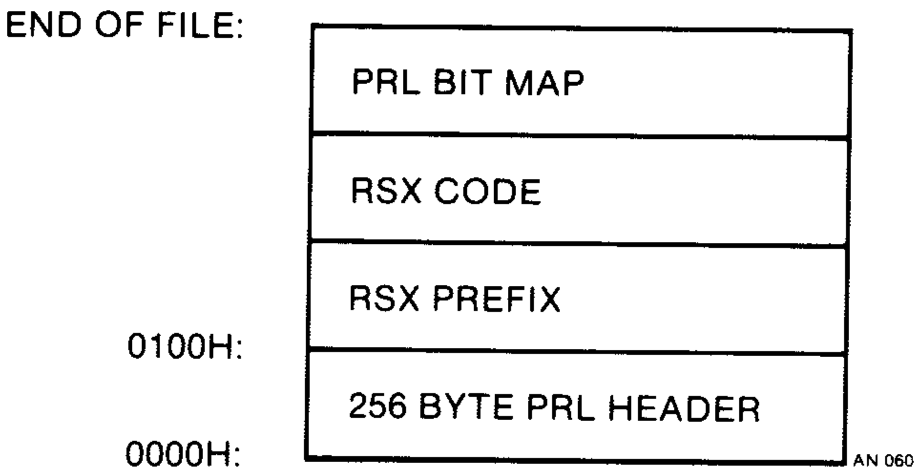

Figure 1-8. RSX File Format

RSX files begin with a one page PRL header that specifies the total size of the RSX prefix and code sections. The PRL bit map is a string of bits identifying those bytes in the RSX prefix and code sections that require relocation. The PRL format is described in detail in Appendix B. Note that the PRL header and bit map are removed when an RSX is loaded into memory. They are only used by the LOADER module to load the RSX.

The RSX prefix is a standard data structure that the LOADER module uses to manage RSXs (see Section 4.4). Included in this data structure are jump instructions to the previous and next RSX in memory, and two flags. The LOADER module initializes and updates these 'ump instructions to maintain the link from location 6 of Page Zero to the BDOS entry point. The RSX flags are the Remove flag and the Nonbanked flag. The Remove flag controls RSX removal from memory. The CCP tests this flag to determine whether or not it should remove the RSX from memory at system warm start. The nonbanked flag identifies RSXs that are loaded only in nonbanked CP/M 3 systems. For example, the CP/M 3 RSX, DIRLBL.RSX, is a nonbanked RSX. It provides BDOS Function 100, Set Directory Label, support for nonbanked systems only. Banked systems support this function in the BDOS.

The RSX code section contains the main body of the RSX. This section always begins with code to intercept the BDOS function that is supported by the RSX. Nonintercepted functions are passed to the next RSX in memory. This section can also include initialization and termination code that transient programs can call with BDOS Function 60.

When the CCP gains control after a system warm start, it removes any RSXs in memory that have the Remove flag set to OFFH. All other RSXs remain active in memory. Setting an RSX's Remove flag to OFFH indicates that the RSX is not active and it can be removed. Note that if an RSX marked for removal is not the lowest active RSX in memory, it still occupies memory after removal. Although the removed RSX cannot be executed, its space is returned to the TPA only when all the lower RSXs are removed.

There 'is one special case where the CCP does not remove an RSX with the Remove flag set to OFFH following warm start. This case occurs on warm starts following the load of an empty file with attached RSXS. This exception allows an RSX with the Remove flag set to be loaded into i-nemory before a transient program. The transient program can then access the RSX during execution. After the transient program terminates, however, the CCP removes the RSX from the system environment.

As an example of RSX operation, here is a description of the operation of the GET utility. The GET.COM command file has an attached RSX. The LOADER moves this RSX to the top of the TPA when it loads the GET.COM command file. The GET utility performs necessary initializations which include opening the ASCII file specified in the GET command line. It also makes a BDOS Function 60 call to initialize the GET.RSX. At this point, the GET utility terminates. Subsequently, the GET.RSX intercepts all console input calls and returns characters from the file specified in the GET command line. It continues this action until it reads end-of-file. At this point, it sets its Remove flag in the RSX prefix, and stops intercepting console input. On the following warm boot, the CCP removes the RSX from memory.

1.6.5 SUBMIT Operation

A SUBMIT command line has the following syntax:

SUBMIT <filespec> <parameters>

If the CCP identifies a command as a submit file, it automatically inserts the SUBMIT keyword into the command line as described in Section 1.6.2.

When the SUBMIT utility begins execution, it opens and reads the file specified by <filespec> and creates a temporary submit file of type $$$ on the system's temporary file drive. GENCPM initializes the temporary file drive to the CCP's current default drive. The SETDEF utility can set the temporary file drive to a specific drive. As it creates the temporary file, SUBMIT performs the parameter substitutions requested by the <parameters> subfield of the SUBMIT command line. See the CP/M Plus (CP/M Version 3) Operating System User's Guide for a detailed description of this process.

After SUBMIT creates the temporary submit file, its operation is similar to that of the GET utility described in Section 1.6.4. The SUBMIT command file also has an attached RSX that performs console input redirection from a file. However, the SUBMIT RSX expands upon the simpler facilities provided by the GET RSX. Command lines in a submit file can be marked to indicate whether they are program or CCP input. Furthermore, if a program exhausts all its program input, the next SUBMIT command is a CCP command, the SUBMIT RSX temporarily reverts to console input. Redirected input from the submit file resumes when the program terminates.

Because CP/M 3's submit facility is implemented with RSXS, submit files can be nested. That is) a submit file can contain additional SUBMIT or GET commands. Similarly, a GET command can specify a file that contains GET or SUBMIT commands. For example, when a SUBMIT command is encountered in a submit file, a new SUBMIT RSX is created below the current RSX. The new RSX handles console input until it reads end-of-file on its temporary submit file. At this point, control reverts to the previous SUBMIT RSX.

1.7 System Control Block

The System Control Block, SCB, is a 100 byte CP/M 3 data structure that resides in the BDOS system component. The SCB contains internal BDOS flags and data, CCP flags and data, and other system information such as console characteristics and the current date and time. The BDOS, BIOS, CCP system components as well as CP/M 3 utilities and RSXs reference SCB fields. BDOS Function 49, Get/Set System Control Block, provides access to the SCB fields for transient programs, RSXS, and the CCP.

However, use caution when you access the SCB through Function 49 for two reasons. First, the SCB is a CP/M 3 data structure. Digital Research's multi-user operating system, MP/M, does not support BDOS Function 49. Programs that access the SCB can run only on CP/M 3. Secondly, the SCB contains critical system parameters that reflect the current state of the operating system. If a program modifies these parameters illegally, the operating system might crash. However, for application writers who are writing system-oriented applications, access to the SCB variables might prove valuable.

For example, the CCP default drive and current user number are maintained in the System Control Block. This information is displayed in the system prompt. If a transient program changes the current disk or user number by making an explicit BDOS call, the System Control Block values are not changed. They continue to reflect the state of the system when the transient program was loaded. For compatibility with CP/M Version 2, the current disk and user number are also maintained in location 0004H of Page Zero. The high-order nibble contains the user number, and the low-order nibble contains the drive.

Refer to the description of BDOS Function 49 in Section 2.5 for more information on the System Control Block. The SCB fields are also discussed in Appendix A.

End of Section 1

Section 2

The BDOS System Interface

This section describes the operating system services available to a transient program through the BDOS module of CP/M 3. The section begins by defining how a transient program calls BDOS functions, then discusses serial I/O for console, list and auxiliary devices, the file system, and Page Zero initialization.

2.1 BDOS Calling Conventions

CP/M 3 uses a standard convention for BDOS function calls. On entry to the BDOS, register C contains the BDOS function number, and register pair DE contains a byte or word value or an information address. BDOS functions return single-byte values in register A, and double-byte values in register pair HL. In addition, they return with register A equal to L, and register H equal to B. If a transient program makes a BDOS call to a nonsupported function number in the range of 0 to 127, the BDOS returns with register pair HL set to OFFFFH. For compatibility with MP/M, the BDOS returns with register pair HL set to OOOOH on nonsupported function numbers in the range of 128 to 255. Note that CP/M 2 returns with HL set to zero on all invalid function calls. CP/M 3's register passing conventions for BDOS function calls are consistent with the conventions used by the Intel PL/M systems programming language.

When a transient program makes a BDOS function call, the BDOS does not restore registers to their entry values before returning to the calling program. The responsibility for saving and restoring any critical re 'ster values rests with the calling program.

When the CCP loads a transient program, the LOADER module sets the stack pointer to a 16 level stack, and then pushes the address OOOOH onto the stack. Thus, an immediate return to the system is equivalent to a jump to OOOOH. However, most transient programs set up their own stack, and terminate execution by making a BDOS System Reset call (Function 0) or by jumping to location OOOOH.

The following example illustrates how a transient program calls a BDOS function. This program reads characters continuously until it encounters an asterisk. Then it termmates execution by returning to the system.

```
bdos     equ    0005h    ;BDOS entry Point in Page Zero
conin    equ    1        ;BDOS console input function
;
         org    100h     ;Base of Transient Program Area
nextc:   mvii   c,conin
         call   bdos     ;Return character in A
         cpi    '*'      ;End of Processing?
         jnz    nextc    ;Loop if not
         ret             ;Terminate Program
         end
```

2.2 BDOS Serial Device I/O

Under CP/M 3, serial device I/O is simply input to and output from simple devices such as consoles, line printers, and communications devices. These physical devices can be assigned the logical device names defined below:

CONIN:	logical console input device
CONOUT:	logical console output device
AUXIN:	logical auxiliary input device
AUXOUT:	logical auxiliary output device
LST:	logical list output device

If your system supports the BIOS DEVTBL function, the CP/M 3 DEVICE utility can display and change the assignment of logical devices to physical devices. DEVICE can also display the names and attributes of physical devices supported on your system. If your system does not support the DEVTBL entry point, then the logical to physical device assignments are fixed by the BIOS.

In general, BDOS serial I/O functions read and write an individual ASCII character, or character string to and from these devices, or test the device's ready status. For these BDOS functions, a string of characters is defined as zero to N characters terminated by a delimiter. A block of characters is defined as zero to N characters where N is specified by a word count field. The maximum value of N in both cases is limited only by available memory. The following list summarizes BDOS serial device I/O functions.

Read a character from CONIN:
Read a character buffer from CONIN:
Write a character to CONOUT:
Write a string of characters to CONOUT:
Write a block of characters to CONOUT:
Read a character from AUXIN:
Write a character to AUXOUT:
Write a character to LST:
Write a block of characters to LST:
Interrogate CONIN:, AUXIN:, AUXOUT: ready

CP/M 3 cannot run unless CONIN: and CONOUT: are assigned to a physical console. The remaining logical devices can remain unassigned. If a logical output device is not assigned to a physical device, an output BDOS call to the logical device performs no action. If a logical input device is not assigned to a physical device, an input BDOS call to the logical device typically returns a CTRL-Z (lAH), which indicates end-of-file. Note that these actions depend on your system's BIOS implementation.

2.2.1 BDOS Console I/O

Because a transient program's main interaction with its user is through the console, the BDOS supports many console I/O functions. Console PO functions can be divided into four categories: basic console I/O, direct console I/O, buffered console input, and special console functions. Using the basic console I/O functions, programs can access the console device for simple input and output. The basic console I/O functions are:

1. Console Input - Inputs a single character
2. Console Output - Outputs a single character
9. Print String - Outputs a string of characters
11. Console Status - Signals if a character is ready for input
111. Print Block - Outputs a block of characters

The input function echoes the character to the console so that the user can identify the typed character. The output functions expand tabs in columns of eight characters, The basic I/O functions also monitor the console to stop and start console output scroll at the user's request. To provide this support, the console output functions make internal status checks for an input character before writing a character to the output device. The console input and console status functions also check the input character. If the user types a CTRL-S, these functions make an additional Blos console input call. This input call suspends execution until a character is typed. If the typed character is not a CTRL-Q, an additional BIOS console input call is made. Execution and console scrolling resume when the user types a CTRL-Q.

When the BDOS is suspended because of a typed CTRL-S, it scans input for three special characters: CTRL-Q, CTRL-C, and CTRL-P. If the user types any other character, the BDOS echoes a bell character, CTRL-G, to the console, discards the input character, and continues the scan. If the user types a CTRL-C, the BDOS executes a warm start which terminates the calling program. If the user types a CTRL-P, the BDOS toggles the printer echo switch. The printer echo switch controls whether console output is automatically echoed to the list device, LST:. The BDOS signals when it turns on printer echo by sending a bell character to the console.

All basic console I/O functions discard any CTRL-Q or CTRL-P character that is not preceded by a CTRL-S character. Thus, BDOS function 1 cannot read a CTRL-S, CTRL-Q, or CTRL-P character. Furthermore, these characters are invisible to the console status function.

The second category of console I/O is direct console I/O. BDOS function 6 can proveide direct console I/O in situations where unadorned console I/O is required. Function 6 actually consists of several sub-functions that support direct console input, output, and status checks. The BDOS does not filter out special characters during direct console I/O. The direct output sub-function does not expand tabs, and the direct input sub-function does not echo typed characters to the console.

The third category of console I/O accepts edited input from the console. The only function in this category, Function 10, Read Buffer Input, reads an input line from a buffer and recognizes certain control characters that edit the input. As an option, the line to be edited can be initialized by the calling program.

In the nonbanked version of CP/M 3, editing within the buffer is restricted to the last character on the line. That. is, to edit a character embedded in the line, the user must delete all characters that follow the erroneous character, correct the error, and then retype the remainder of the line. The banked version of CP/M 3 supports complete line editing in which characters can be deleted and inserted anywhere in the line. In addition', the banked version can also recall the previously entered line.

Function 10 also filters input for certain control characters. If the user types a CTRL-C as the first character in the line, Function 10 terminates the calling program by branching to the BIOS warm start entry point. A CTRL-C in any other position is simply echoed at the console. Function 10 also watches for a CTRL-P keystroke, and if it finds one at any position in the command line, it toggles the printer echo switch. Function 10 does not filter CTRL-S and CTRL-Q characters, but accepts them as normal input. In general, all control characters that Function 10 does not recognize as editing control characters, it accepts as input characters. Function 10 identifies a control character with a leading caret, ', when it echoes the control character to the console. Thus, CTRL-C appears as 'C in a Function 10 command line on the screen.

The final category of console I/O functions includes special functions that modify the behavior of other console functions. These functions are:

109. Get/Set Console Mode
110. Get/Set Output Delimiter

Function 110 can get or set the current delimiter for Function 9, Print String. The delimiter is $, when a transient program begins execution. Function 109 gets or sets a 16-bit system variable called the Console Mode. The following list describes the bits of the Console Mode variable and their functions:

bit 0 If this bit is set, Function 11 returns true only if a CTRL-C is typed at the console. Programs that make repeated console status calls to test if execution should be interrupted, can set this bit to interrupt on CTRL-C only. The CCP DIR and TYPE built-in commands run in this mode.

bit 1 Setting this bit disables stop and start scroll support for the basic console 110 functions, which comprise the first category of functions described in this section. When this bit is set, Function 1 reads CTRL-S, CTRL-Q, and CTRL-P, and Function 1 1 returns true if the user types these characters. Use this mode in situations where raw console input and edited output is needed. While in this mode, you can use Function 6 for input and input status, and Functions 1, 9, and 111 for output without the possibility of the output functions intercepting input CTRL-S, CTRL-Q, or CTRL-P characters.

bit 2 : Setting this bit disables tab expansion and printer echo support for Functions 2, 9, and 111. Use this mode when non-edited output is required.

bit 3 : This bit disables all CTRL-C intercept action in the BDOS. This mode is useful for programs that must control their own termination.

bits 8 and 9 : The BDOS does not use these bits, but reserves them for the CP/M 3 GET RSX that performs console input redirection from a file. With one exception, these bits determine how the GET RSX responds to a program console status request (Function 6, Function 11, or direct BIOS).

bit 8 = 0, bit 9 = 0 - conditional status
bit 8 = 0, bit 9 = 1 - false status
bit 8 = 1, bit 9 = 0 - true status
bit 8 = 1, bit 9 = 1 - do not perform redirection

In conditional status mode, GET responds false to all status requests except for a status call preceded immediately by another status call. On the second call, GET responds with a true result. Thus, a program that spins on status to wait for a character is signaled that a character is ready on the second call. In addition, a program that makes status calls periodically to see if the user wants to stop is not signaled.

When a transient program begins execution, the Console Mode bits are normally set to zero. However,, the CP/M 3 utility GENCOM can attach an RSX header to a COM file so that when it is loaded, the console mode bits are set differently. This feature allows you to modify a program's console I/O behavior without having to change the program.

2.2.2 Other Serial I/O

The BDOS supports single character output functions for the logical devices LST: and AUXOUT:, an input function for AUXIN:, and status functions for AUXIN: and AUXOUT:. A block output function is also supported for the LST: device. Unlike the console I/O functions, the BDOS does not intercept control characters or expand tabs for these functions. Note that AUXIN: and AUXOUT: replace the READER and PUNCH devices supported by earlier versions of CP/M.

2.3 BDOS File System

Transient programs depend on the BDOS file system to create, update, and maintain disk files. This section describes the capabilities of the BDOS file system in detail. You must understand the general features of CP/M 3 described in Section 1 before you can use the detail presented in this section.

The remaining introductory paragraphs define the four categories of BDOS file functions. This is followed by a review of file naming conventions and disk and file organization. The section then describes the data structure used by the BDOS file, and directory oriented functions: the File Control Block (FCB). Subsequent discussions cover file attributes, user numbers, directory labels and extended File Control Blocks (XFCBs), passwords, date and time stamping, blocking and deblocking, multisector I/O, disk reset and removable media, byte counts, and error handling. These topics are closely related to the BDOS file system. You must be familiar with the contents of Section 2 before attempting to use the BDOS functions described individually in Section 3.

The BDOS file system supports four categories of functions: file access functions, directory functions, drive related functions, and miscellaneous functions. The file access category includes functions to create a file, open an existing file, and close a file. Both the make and open functions activate the file for subsequent access by BDOS file access functions. The BDOS read and write functions are file access functions that operate either sequentially or randomly by record position. They transfer data in units of 128 bytes, which is the basic record size of the file system. The close function makes any necessary updates to the directory to permanently record the status of an activated file.

BDOS directory functions operate on existing file entries in a drive's directory. This category includes functions to search for one or more files, delete one or more files, truncate a file, rename a file, set file attributes, assign a password to a file, and compute the size of a file. The search and delete functions are the only BDOS functions that support ambiguous file references. All other directory and fi e related functions require a specific file reference.

The BDOS drive-related category includes functions that select the default drive, compute a drive's free space, interrogate drive status, and assign a directory label to a drive. A drive's directory label controls whether or not CP/M 3 enforces file password protection, or stamps files with the date and time. Note that the nonbanked version of CP/M 3 does not support file passwords.

The miscellaneous category includes functions to set the current DMA address, access and update the current user number, chain to a new program, and flush internal blocking/deblocking buffers. Also included are functions that set the BDOS multi-sector count, and the BDOS error mode. The BDOS multi-sector count determines the number of 128-byte records to be processed by BDOS read and write functions. It can range from 1 to 128. The BDOS error mode determines how the BDOS file system handles certain classes of errors.

Also included in the miscellaneous category are functions that call the BIOS directly, set a program return code, and parse filenames. If the LOADER RSX is resident in memory, programs can also make a BDOS function call to load an overlay. Another miscellaneous function accesses system variables in the System Control Block.

The following list summarizes the operations performed by the BDOS file system-

Disk System Reset
Drive Selection
File Creation
File Open
File Close
Directory Search
File Delete
File Rename
Random or Sequential Read
Random or Sequential Write
Interrogate Selected Disks
Set DMA Address
Set/Reset File Attributes
Reset Drive
Set BDOS Multi-Sector Count
Set BDOS Error Mode
Get Disk Free Space
Chain to Program
Flush Buffers
Get/Set System Control Block
Call BIOS
Load Overlay
Call RSX
Truncate File
Set Directory Label
Get File's Date Stamps and Password Mode
Write File XFCB
Set/Get Date and Time
Set Default Password
Return CP/M 3 Serial Number
Get/Set Program Return Code
Parse Filename

2.3.1 File Naming Conventions

Under CP/M 3, a file specification consists of four parts: the drive specifier, the filename field, the filetype field, and the file password field. The general format for a command line file specification is shown below:

{d:}filename{.typ}{;password}

The drive specifier field specifies the drive where the file is located. The filename and type fields identify the file. The password field specifies the password if a file is password protected.

-ne drive, type, and password fields are optional, and the delimiters :.; are required only when specifying their associated field. The drive specifier can be assigned a letter from A to P where the actual drive letters supported on a given system are determined by the BIOS implementation. When the drive letter is not specified, the current default drive is assumed.

The filename and password fields can contain one to eight non-delimiter characters. The filetype field can contain one to three non-delimiter characters. All three fields are padded with blanks, if necessary. Omitting the optional type or password fields implies a field specification of all blanks.

The CCP calls BDOS Function 152, Parse Filename, to parse file specifications from a command line. Function 152 recognizes certain ASCII characters as valid delimiters when it parses a file from a command line. The valid delimiters are shown in Table 2-1.

Table 2-1. Valid Filenamc Delimiters

<table>
<tr><th>ASCII</th><th>HEX EQUIVALENT</th></tr>
<tr><td>null</td><td>00</td></tr>
<tr><td>space</td><td>20</td></tr>
<tr><td>return</td><td>0D</td></tr>
<tr><td>tab</td><td>09</td></tr>
<tr><td>:</td><td>3A</td></tr>
<tr><td>.</td><td>2E</td></tr>
<tr><td>;</td><td>3B</td></tr>
<tr><td>=</td><td>3D</td></tr>
<tr><td>,</td><td>2C</td></tr>
<tr><td>[</td><td>5B</td></tr>
<tr><td>]</td><td>5D</td></tr>
<tr><td><</td><td>3C</td></tr>
<tr><td>></td><td>3E</td></tr>
<tr><td>|</td><td>7C</td></tr>
</table>

Function 152 also excludes all control characters from the file fields, and translates all lower-case letters to upper-case.

Avoid using parentheses and the backslash character, \, in the filename and filetype fields because they are commonly used delimiters. Use asterisk and question mark characters, * and ?, only to make an ambiguous file reference. When Function 152 encounters an * in a filename or filetype field, it pads the remainder of the field with question marks. For example, a filename of X*.* is parsed to X???????.???. The BDOS search and delete functions treat a ? in the filename and type fields as follows: A ? in any position matches the corresponding field of any directory entry belonging to the current user number. Thus, a search operation for X???????.??? finds all the current user files on the directory beginning in X. Most other file related BDOS functions treat the presence of a ? in the filename or type field as an error.

It is not mandatory to follow the file naming conventions of CP/M 3 when you create or rename a file with BDOS functions. However, the conventions must be used if the file is to be accessed from a command line. For example, the CCP cannot locate a command file in the directory if its filename or type field contains a lower-case letter.

As a general rule, the filetype field names the generic category of a particular file, while the filename distinguishes individual files in each category. Although they are generally arbitrary, the following list of filetypes names some of the generic categories that have been established.

ASM	Assembler Source	PLI	PL/I Source File
PRN	Printer Listing	REL	Relocatable Module
HEX	Hex Machine Code	TEX	TEX Formatter Source
BAS	Basic Source File	BAK	ED Source Backup
INT	Intermediate File	SYM	SID Symbol File
COM	Command File	$$$	Temporary File
PRL	Page Relocatable	DAT	Data File
SPR	Sys. Page Reloc.	SYS	System File

2.3.2 Disk and File Organization

The BDOS file system can support from one to sixteen logical drives. The maximum file size supported on a drive is 32 megabytes. The maximum capacity of a drive is determined by the data block size specified for the drive in the BIOS. The data block size is the basic unit in which the BDOS allocates disk space to files.

Table 2-2 displays the relationship between data block size and drive capacity.

Table 2-2. Logical Drive Capacity

Data Block Size	Maximum Drive Capacity
1K	256 Kilobytes
2K	64 Megabytes
4K	128 Megabytes
8K	256 Megabytes
16K	512 Megabytes

Logical drives are divided into two regions: a directory area and a data area. The directory area contains from one to sixteen blocks located at the beginning of the drive. The actual number is set in the BIOS. This area contains entries that define which files exist on the drive. The directory entries corresponding to a particular file define those data blocks in the drive's data area that belong to the file. These data blocks contain the file's records. The directory area is logically subdivided into sixteen independent directories identified as user 0 through 15. Each independent directory shares the actual directory area on the drive. However, a file's directory entries cannot exist under more than one user number. In general, only files belonging to the current user number are visible in the directory.

Each disk file consists of a set of up to 262,144 128-byte records. Each record in a file is identified by its position in the file. This position is called the record's random record number. If a file is created sequentially, the first record has a position of zero, while the last record has a position one less than the number of records in the file. Such a file can be read sequentially in record position order beginning at record zero, or randomly by record position. Conversely, if a file is created randomly, records are added to the file by specified position. A file created in this way is called sparse if positions exist within the file where a record has not been written.

The BDOS automatically allocates data blocks to a file to contain its records on the basis of the record positions consumed. Thus, a sparse file that contains two records, one at position zero, the other at position 262,143, consumes only two data blocks in the data area. Sparse files can only be created and accessed randomly, not sequentially. Note that any data block allocated to a file is permanently allocated to the file until the file is deleted or truncated. These are the only mechanisms supported by the BDOS for releasing data blocks belonging to a file.

Source files under CP/M 3 are treated as a sequence of ASCII characters, where each line of the source file is followed by a carriage return line-feed sequence, ODH followed by OAH. Thus a single 128-byte record could contain several lines of source text. The end of an ASCII file is denoted by a CTRL-Z character, 1AH, or a real end of file, returned by the BDOS read operation. CTRL-Z characters embedded within machine code files such as COM files are ignored. The actual end-of-file condition returned by the BDOS is used to terminate read operations.

2.3.3 File Control Block Definition

The File Control Block, FCB, is a data structure that is set up and initialized by a transient program, and then used by any BDOS file access and directory functions called by the transient program. Thus the FCB is an important channel for information exchange between the BDOS and a transient program. For example, when a program opens a file, and subsequently accesses it with BDOS read and write record functions, the BDOS file system maintains the current file state and position within the program's FCB. Some BDOS functions use certain fields in the FCB for invoking special options. Other BDOS functions use the FCB to return data to the calling program. In addition, all BDOS random I/O functions specify the random record number with a 3-byte field at the end of the FCB.

When a transient program makes a file access or directory BDOS function call, register pair DE must address an FCB. The length of the FCB data area depends on the BDOS function. For most functions, the required length is 33 bytes. For random I/O functions, the Truncate File function, and the Compute File Size function, the FCB length must be 36 bytes. The FCB format is shown on the next page.

dr	f1	f2	...	f8	t1	t2	t3	ex	s1	s2	rc	d0	...	dn	cr	r0	r1	r2
00	01	02	...	08	09	10	11	12	13	14	15	16	...	31	32	33	34	35

where

dr drive code (O - 16)
0 = > use default drive for file
1 = > auto disk select drive A,
2 = > auto disk select drive B,

16 = > auto disk select drive P.

fl ... f8 contain the filename in ASCII
upper-case, with high bit = 0.
f1'..., f8' denote the high-
order bit of these positions,
and are file attribute bits.

tl,t2,t3 contain the filetype in ASCII
upper-case, with high bit = 0.
tl') t2'1 and t3' denote the
high bit of these positions,
and are file attribute bits.
tl' = 1 = > Read/Only file
t2' = 1 = > System file
t3' = 1 = > File has been archived

ex contains the current extent number,
usually set to 0 by the calling program,
but can range 0 - 31 during file I/O

sl reserved for internal system use

s2 reserved for internal system use

rc record count for extent "ex"
takes on values from 0 - 255
(values greater than 128 imply
record count equals 128)

d0 ... dn filled-in by CP/M 3, reserved for system use

cr current record to read or write in a sequential file operation, normally set to zero by the calling program when a file is opened or created

r0,,rl,r2 optional random record number in the range 0-262,143 (O - 3FFFFH). r0,rl,r2 constitute a 18 bit value with low byte rO, middle byte rl, and high byte r2.

For BDOS directory functions, the calling program must initialize bytes 0 through 11 of the FCB before issuing the function call. The Set Directory Label and Write File XFCB functions also require the calling program to initialize byte 12. The Rename File function requires the calling program to place the new filename and type in bytes 17 through 27.

BDOS open or make function calls require the calling program to intialize bytes 0 through 12 of the FCB before making the call. Usually, byte 12 is set to zero. In
if the file is to be processed from the beginning using sequential read or
addition, 1
write functions, byte 32, cr, must be zeroed.

After an FCB is activated by an open or make operation, a program does not have to modify the FCB to perform sequential read or write operations. In fact, bytes 0 through 31 of an activated FCB should not be modified. However, random I/O functions require that a program set bytes 33 through 35 to the requested random record number prior to making the function call.

File directory entries maintained in the directory area of each disk have the same format as FCBS, excluding bytes 32 through 35, except for byte 0 which contains the file's user number. Both the Open File and Make File functions bring these entries, excluding byte 0, into memory in the FCB specified by the calling program. All read and write operations on a file must specify an FCB activated in this manner.

The BDOS updates the memory copy of the FCB during file processing to maintain the current position within the file. During file write operations, the BDOS updates the memory copy of the FCB to record the allocation of data to the file, and at the termination of file processing, the Close File function permanently records this information on disk. Note that data allocated to a file during file write operations is not completely recorded in the directory until the calling program issues a Close File call. Therefore,, a program that creates or modifies files must close the files at the end of any write processing. Otherwise, data might be lost.

The BDOS Search and Delete functions support multiple or ambiguous file references. In general, a question mark in the filename, filetype, or extent field matches any value in the corresponding positions of directory FCBs during a directory search operation. The BDOS search functions also recognize a question mark in the drive code field, and if specified, they return all directory entries on the disk regardless of user number, including empty entries. A directory FCB that begins with E5H is an empty directory entry.

2.3.4 File Attributes

The high-order bits of the FCB filename, fl',...,f8', and filetype, tl',t2',t3', fields are called attribute bits. Attributes bits are 1 bit Boolean fields where 1 indicates on or true, and 0 indicates off or false. Attribute bits indicate two kinds of attributes within the file system: file attributes and interface attributes.

The file attribute bits, fl ...... f4' and tl',t2',t3', can indicate that a file has a defined file attribute. These bits are recorded in a file's directory FCBS. File attributes can be set or reset only by the BDOS Set File Attributes function. When the BDOS Make File function creates a file, it initializes all file attributes to zero. A program can interrogate file attributes in an FCB activated by the BDOS Open File function, or in directory FCBs returned by the BDOS Search For First and Search For Next functions.

Note: the BDOS file system ignores file attribute bits when it attempts to locate a file in the directory.

The file system defines the file attribute bits, tl',t2',t3', as follows:

tl': Read-Only attribute - The file system prevents write operations to a file with the read-only attribute set.

t2': System attribute - This attribute, if set, identifies the file as a CP/M 3 system file. System files are not usually displayed by the CP/M 3 DIR command. In addition, user-zero system files can be accessed on a read-only basis from other user numbers.

t3': Archive attribute - This attribute is designed for user written archive programs. When an archive program copies a file to backup storage, it sets the archive attribute of the copied files. The file system automatically resets the archive attribute of a directory FCB that has been issued a write command. The archive program can test this attribute in each of the file's directory FCBs via the BDOS Search and Search Next functions. If all directory FCBs have the archive attribute set, it indicates that the file has not been modified since the previous archives Note that the CP/M 3 PIP utility supports file archival.

Attributes fl' through f4' are available for definition by the user.

The interface attributes are indicated by bits f5' through f8' and cannot be used as file attributes. Interface attributes f5' and f6' can request options for BDOS Make File, Close File, Delete File, and Set File Attributes functions. Table 2-3 defines options indicated by the f5' and f6' interface attribute bits for these functions.

Table 2-3. BDOS Interface Attributes

BDOS Function	Interface Attribute Definition
16. Close File	f5' 1 Partial Close
19. Delete File	f5' 1 Delete file XFCBs only
22. Make File	f6' = 1 Assign password to file
30. Set File Attributes	f6' = 1 Set file byte count

Section 3 discusses each interface attribute in detail in the definitions of the above functions. Attributes f5' and f6' are always reset when control is returned to the calling program. Interface attributes f7' and f8' are reserved for internal use by the BDOS file system.

2.3.5 User Number Conventions

The CP/M 3 User facility divides each drive directory into sixteen logically independent directories, designated as user 0 through user 15. Physically, all user directories share the directory area of a drive. In most other aspects, however, they are independent. For example, files with the same name can exist on different user numbers of the same drive with no conflict. However, a single file cannot reside under more than one user number.

Only one user number is active for a program at one time, and the current user number applies to all drives on the system. Furthermore, the FCB format does not contain any field that can be used to override the current user number. As a result, all file and directory operations reference directories associated with the current user number. However, it is possible for a program to access files on different user numbers; this can be accomplished by setting the user number to the file's user number with the BDOS Set User function before making the desired BDOS function call for the file. Note that this technique must be used carefully. An error occurs if a program attempts to read or write to a file under a user number different from the user number that was active when the file was opened.

When the CCP loads and executes a transient program, it initializes the user number to the value displayed in the system prompt. If the system prompt does not display a user number, user zero is implied. A transient program can change its user number by making a BDOS Set User function call. Changing the user number in this way does not affect the CCP's user number displayed in the system prompt. When the transient program terminates, the CCP's user number is restored. However, an option of the BDOS Program Chain command allows a program to pass its current user number and default drive to the chained program.

User 0 has special properties under CP/M 3. When the current user number is not equal to zero, and if a requested file is not present under the current user number, the file system automatically attempts to open the file under user zero. If the file exists under user zero, and if it has the system attribute, t2', set, the file is opened from user zero. Note, however, that files opened in this way cannot be written to; they are available only for read access. This procedure allows utilities that may include overlays and any other commonly accessed files to be placed on user zero, but also be available for access from other user numbers. As a result, commonly needed utilities need not be copied to all user numbers on a directory, and you can control which user zero files are directly accessible from other user numbers.

2.3.6 Directory Labels and XFCBs

The BDOS file system includes two special types of FCBS: the XFCB and the Directory Label. The XFCB is an extended FCB that optionally can be associated with a file in the directory. If present, it contains the file's password. Note that password protected files and XFCBs are supported only in the banked version of CP/M 3. The format of the XFCB follows.

DR	FILE	TYPE	PM	S1	S2	RC	PASSWORD	RESERVED
00	01 . . .	09 . .	12	13	14	15	16	24 . . . . . .

AN 069

Figure 2-1. XFCB Format

dr	-	drive code (O - 16)
file	-	filename field
type	-	filetype field
PM	-	password mode bit 7 - Read mode bit 6 - Write mode bit 5 - Delete mode ** - bit references are right to left, relative to 0
sl,s2, rc	-	reserved for system use
password	-	8-byte password field (encrypted)
reserved	-	8-byte reserved area

An XFCB can be created only on a drive that has a directory label, and only if the directory label has activated password protection. For drives in this state, an XFCB can be created for a file in two ways: by the BDOS Make function or by the BDOS Write File XFCB function. The BDOS Make function creates an XFCB if the calling program requests that a password be assigned to the created file. The BDOS Write File XFCB function can be used to assign a password to an existing file. Note that in the directory, an XFCB is identified by a drive byte value, byte 0 in the FCB, equal to 16 + N, where N equals the user number.

For its drive, the directory label specifies if file password support is to be activated, and if date and time stamping for files is to be performed. The format of the Directory Label follows.

DR	NAME	TYPE	D1	S1	S2	RC	PASSWORD	TS1	TS2
00	01 . .	09 . .	12	13	14	15	16	24 .	28 .

AN 070

Figure 2-2. Directory Label Format

dr	-	drive code (0 - 16)
name	-	Directory Label name
type	-	Directory Label type
dl	-	Directory Label data byte bit 7 - require passwords for password protected files bit 6 - perform access time stamping bit 5 - perform update time stamping bit 4 -perform create time stamping bit 0 -Directory Label exists ** - bit references are right to left, relative to 0
sl,s2,rc	-	n/a
password	-	8-byte password field (encrypted)
tsl	-	4-byte creation time stamp field
ts2	-	4-byte update time stamp field

Only one Directory Label can exist in a drive's directory. The Directory Label name and type fields are not used to search for a Directory Label; they can be used to identify a disk. A Directory Label can be created, or its fields can be updated by BDOS function 100, Set Directory Label. This function can also assign a Directory Label a password. The Directory Label password, if assigned, cannot be circumvented, whereas file password protection is an option controlled by the Directory Label. Thus, access to the Directory Label password provides a kind of super-user status on that drive.

The nonbanked version of CP/M 3 does not support file passwords. However, it does provide password protection of directory labels. The CP/M 3 RSX, DIRLBL.RSX, which implements BDOS Function 100 in the nonbanked version of CP/M 3, provides this support.

The BDOS file system has no function to read the Directory Label FCB directly. However, the Directory Label data byte can be read directly with the BDOS Function 101, Return Directory Label. In addition, the BDOS Search functions, with a ? in the FCB drive byte, can be used to find the Directory Label on the default drive. In the directory, the Directory Label is identified by a drive byte value, byte 0 in the FCB, equal to 32, 20H.

2.3.7 File Passwords

Only the banked version of CP/M 3 supports file passwords. In the nonbanked version, all BDOS functions with password related options operate the same way the banked version does when passwords are not enabled.

Files can be assigned passwords in two ways: by the Make File function or by the Write File XFCB function. A file's password can also be changed by the Write File XFCB function if the original password is supplied.

Password protection is provided in one of three modes. Table 2-4 shows the difference in access level allowed to BDOS functions when the password is not supplied.

Table 2-4. Password Protection Modes

Password Mode	Access level allowed wben the Password is not supplied.
1. Read	The file cannot be read.
2. Write	The file can be read, but not modified.
3. Delete	The file can be modified, but not deleted.

If a file is password protected in Read mode, the password must be supplied to open the file. A file protected in Write mode cannot be written to without the password. A file protected in Delete mode allows read and write access, but the user must specify the password to delete the file, rename the file, or to modify the file's attributes. Thus, password protection in mode 1 implies mode 2 and 3 protection, and mode 2 protection implies mode 3 protection. All three modes require the user to specify the password to delete the file, rename the file, or to modify the file's attributes.

If the correct password is supplied, or if password protection is disabled by the Directory Label, then access to the BDOS functions is the same as for a file that is not password protected. In addition, the Search For First and Search For Next functions are not affected by file passwords.

Table 2-5 lists the BDOS functions that test for password.

Table 2-5. BDOS Functions That Test For Password

15.	Open File
19.	Delete File
23.	Rename File
30.	Set File Attributes
99.	Truncate File
100.	Set Directory Label
103.	Write File XFCB

File passwords are eight bytes in length. They are maintained in the XFCB Directory Label in encrypted form. To make a BDOS function call for a file that requires a password, a program must place the password in the first eight bytes of the current DMA, or specify it with the BDOS function, Set Default Password, prior to making the function call.
Note: the BDOS keeps an assigned default password value until it is replaced with a new assigned value.

2.3.8 File Date and Time Stamps

The CP/M 3 File System uses a special type of directory entry called an SFCB to record date and time stamps for files. When a directory has been initialized for date and time stamping, SFCBs reside in every fourth position of the directory. Each SFCB maintains the date and time stamps for the previous three directory entries as shown in Figure 2-3.

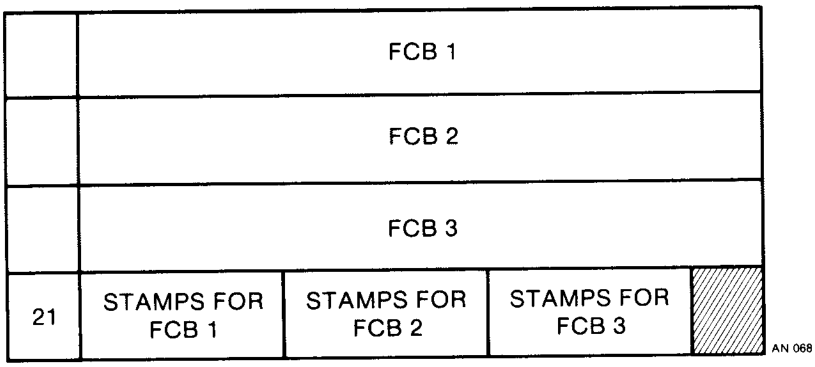

Figure 2-3. Directory Record with SFCB

This figure shows a directory record that contains an SFCB. Directory records consist of four directory entries, each 32 bytes long. SFCBs always occupy the last position of a directory record.

The SFCB directory item contains five fields. The first field is one byte long and contains the value 21H. This value identifies the SFCB in the directory. The next three fields, the SFCB subfields, contain the date and time stamps for their corresponding FCB entries in the directory record. These fields are 10 bytes long. The last byte of the SFCB is reserved for system use. The format of the SFCB subfields is shown in Table 2-6.

Table 2-6. SFCB Subfields Format

Offset in Bytes	SFCB Subfield Contents
0 - 3	Create or Access Date and Time Stamp field
4 - 7	Update Date and Time Stamp field
8	Password mode field
9	Reserved

An SFCB subfield contains valid information only if its corresponding FCB in the directory record is an extent zero FCB. This FCB is a file's first directory entry. For password protected files, the SFCB subfield also contains the password mode of the file. This field is zero for files that are not password protected. The BDOS Search and Search Next functions can be used to access SFCBs directly. In addition, BDOS Function 102 can return the file date and time stamps and password mode for a specified file. Refer to Section 3, function 102, for a description of the format of a date and time stamp field.

CP/M 3 supports three types of file stamping: create, access, and update. Create stamps record when the file was created, access stamps record when the file was last opened, and update stamps record the last time the file was modified. Create and access stamps share the same field. As a result, file access stamps overwrite any create stamps.

The CP/M 3 utility, INITDIR, initializes a directory for date and time stamping by placing SFCBs in every fourth directory entry. Date and time stamping is not supported on disks that have not been initialized in this manner. For initialized disks the disks' Directory Label determines the type of date and time stamping supported for files on the drive. If a disk does not have a Directory Label, or if it is Read-Only, or if the disk's Directory Label does not specify date and time stamping, then date and time stamping for files is not performed. Note that the Directory Label is also time stamped, but these stamps are not made in an SFCB. Time stamp fields in the last eight bytes of the Directory Label record when it was created and last updated. Access stamping for Directory Labels is not supported.

The BDOS file system uses the CP/M 3 system date and time when it records a date and time stamp. This value is maintained in a field in the System Control Block (SCB). On CP/M 3 systems that support a hardware clock, the BIOS module directly updates the SCB system date and time field. Otherwise, date and time stamps record the last initialized value for the system date and time. The CP/M 3 DATE utility can be used to set the system date and time.

2.3.9 Record Blocking and Deblocking

Under CP/M 3, the logical record size for disk I/O is 128 bytes. This is the basic unit of data transfer between the operating system and transient programs. However, on disk, the record size is not restricted to 128 bytes. These records, called physical records, can range from 128 bytes to 4K bytes in size. Record blocking and deblocking is required on systems that support drives with physical record sizes larger than 128 bytes.

The process of building up physical records from 128 byte logical records is called record blocking. This process is required in write operations. The reverse process of breaking up physical records into their component 128 byte logical records is called record deblocking. This process is required in read operations. Under CP/M 3, record blocking and deblocking is normally performed by the BDOS.

Record deblocking implies a read-ahead operation. For example, if a transient program makes a BDOS function call to read a logical record that resides at the beginning of a physical record, the entire physical record is read into an internal buffer. Subsequent BDOS read calls for the remaining logical records access the buffer instead of the disk. Conversely, record blocking results in the postponement of physical write operations but only for data write operations. For example, if a ransient program makes a BDOS write call, the logical record is placed in a buffer equal in size to the physical record size. The write operation on the physical record
t
buffer is postponed until the buffer is needed in another I/O operation. Note that under CP/M 3, directory write operations are never postponed.

Postponing physical record write operations has implications for some applications programs. For those programs that involve file updating, it is often critical to guarantee that the state of the file on disk parallels the state of the file in memory after the update operation. This is only an issue on systems where physical write operations are postponed because of record blocking and deblocking. If the system should crash while a physical buffer is pending, data would be lost. To prevent this loss of data, the BDOS Flush Buffers function, function 48, can be called to force the write of any pending physical buffers.

Note: the CCP automatically discards all pending physical data buffers when it receives control following a system warm start. However, the BDOS file system automatically makes a Flush Buffers call in the Close File function. Thus, it is sufficient to close a file to ensure that all pending physical buffers for that file are written to the disk.

2.3.10 Multi-Sector I/O

CP/M 3 can read or write multiple 128-byte records in a single BDOS function call. This process, called multi-sector I/O, is useful primarily in sequential read and write operations, particularly on drives with physical record sizes larger than 128 bytes. In a multi-sector I/O operation, the BDOS file system bypasses, when possible, all intermediate record buffering. Data is transferred directly between the TPA and the drive. In addition, the BDOS informs the BIOS when it is reading or writing multiple physical records in sequence on a drive. The BIOS can use this information to further optimize the I/O operation resulting in even better performance. Thus, the primary objective of multi-sector I/O is to improve sequential I/O performance. The actual improvement obtained, however, depends on the hardware environment of the host system, and the implementation of the BIOS.

The number of records that can be supported with multi-sector I/O ranges from 1 to 128. This value can be set by BDOS function 44, Set multi-sector Count. The multi-sector count is set to one when a transient program begins execution. However, the CP/M 3 LOADER module executes with the multi-sector Count set to 128 unless the available TPA space is less than 16K. In addition, the CP/M 3 PIP utility also sets the multi-sector count to 128 when sufficient buffer space is available. Note that the greatest potential performance increases are obtained when the multi-sector count is set to 128. Of course, this requires a 16K buffer.

The multi-sector count determines the number of operations to be performed by the following BDOS functions:

- o Sequential Read and Write functions
- o Random Read and Write functions including Write Random with Zero Fill

If the multi-sector count is N, calling one of the above functions is equivalent to making N function calls. If a multi-sector I/O operation is interrupted with an error such as reading unwritten data, the file system returns in register H the number of 128-byte records successfully processed.

2.3.11 Disk Reset and Removable Media

The BDOS functions, Disk Reset (function 13) and Reset Drive (function 37) allow a program to control when a disk's directory is to be reinitialized for file operations. This process of initializing a disk's directory is called logging-in the drive. When CP/M 3 is cold started, all drives are in the reset state. Subsequently, as drives are referenced, they are automatically logged-in by the file system. Once logged-in, a drive remains in the logged-in state until it is reset by BDOS function 13 or 37. Following the reset operation, the drive is again automatically logged-in by the file system when it is next used. Note that BDOS functions 13 and 37 have similar effects except that function 13 is directed to all drives on the system. Any combination of drives can be reset with Function 37.

Logging-in a drive consists of several steps. The most important step is the initialization of the drive's allocation vector. The allocation vector records the allocation and deallocation of data blocks to files, as files are created, extended, deleted, and truncated, Another function performed during drive log-in is the initialization of the directory check-sum vector. The file system uses the check-sum vector to detect media changes on a drive. Note that permanent drives, which are drives that do not support media changes, might not have check-sum vectors. If directory hashing has been specified for the drive, a BIOS and GENCPM option, the file system creates a hash table for the directory during log-in.

The primary use of the drive reset functions is to prepare for a media change on a drive. Subsequently, when the drive is accessed by a BDOS function call, the drive is automatically logged-in. Resetting a drive has two important side effects. First of all, any pending blocking/deblocking buffers on the reset drive are discarded. Secondly, any data blocks that have been allocated to files that have not been closed are lost. An application program should close files, particularly files that have been written to, prior to resetting a drive.

Although CP/M 3 automatically relogs in removable media when media changes are detected, the application program should still explicitly reset a drive before prompting the user to change disks.

2.3.12 File Byte Counts

Although the logical record size of CP/M 3 is restricted to 128 bytes, CP/M 3 does provide a mechanism to store and retrieve a byte count for a file. This facility can identify the last byte of the last record of a file. The BDOS Compute File Size function returns the random record number, plus 1, of the last record of a file.

The BDOS Set File Attributes function can set a file's byte count. Conversely, the Open function can return a file's byte count to the cr field of the FCB. The BDOS Search and Search Next functions also return a file's byte count. These functions return the byte count in the sl field of the FCB returned in the current DMA buffer (see BDOS Functions Returned 17 and 26).

Note that the file system does not access or update the byte count value in file read or write operations. However, the BDOS Make File function does set the byte count of a file to zero when it creates a file in the directory.

2.3.13 BDOS Error Handling

The BDOS file system responds to error situations in one of three ways:

Method 1.	It returns to the calling program with return codes in register A, H, and L identifying the error.
Method 2.	It displays an error message on the console, and branches to the BIOS warm start entry point, thereby terminating execution of the calling program.
Method 3.	It displays an error message on the console, and returns to the calling program as in method 1.

The file system handles the majority of errors it detects by method 1. Two examples of this kind of error are the file not found error for the open function and the reading unwritten data error for a read function. More serious errors, such as disk I/O errors, are usually handled by method 2. Errors in this category, called physical and extended errors, can also be reported by methods 1 and 3 under program control.

The BDOS Error Mode, which can exist in three states, determines how the file system handles physical and extended errors. In the default state, the BDOS displays the error message, and terminates the calling program, method 2. In return error mode, the BDOS returns control to the calling program with the error identified in registers A, H, and L, method 1. In return and display mode, the BDOS returns control to the calling program with the error identified in registers A, H, and L, and also displays the error message at the console, method 3. While both return modes protect a program from termination because of a physical or extended error, the return and display mode also allows the calling program to take advantage of the built-in error reporting of the BDOS file system. Physical and extended errors are displayed on the console in the following format:

CP/M Error on d: error message
BDOS function = nn File = filename.typ

where d identifies the drive selected when the error condition is detected; error message identifies the error; nn is the BDOS function number, and filename.typ identifies the file specified by the BDOS function. If the BDOS function did not involve an FCB, the file information is omitted. Note that the second line of the above error message is displayed only in the banked version of CP/M 3 if expanded error message reporting is requested in GENCPM. It is not displayed in the nonbanked version of CP/M 3.

The BDOS physical errors are identified by the following error messages:

o Disk I/O
o Invalid Drive
o Read-Only File
o Read-Only Disk

The Disk I/O error results from an error condition returned to the BDOS from the BIOS module. The file system makes BIOS read and write calls to execute file-related BDOS calls. If the BIOS read or write routine detects an error, it returns an error code to the BDOS resulting in this error.

The Invalid Drive error also results from an error condition returned to the BDOS from the BIOS module. The BDOS makes a BIOS Select Disk call prior to accessing a drive to perform a requested BDOS function. If the BIOS does not support the selected disk, the BDOS returns an error code resulting in this error message.

The Read-Only File error is returned when a program attempts to write to a file that is marked with the Read-Only attribute. It is also returned to a program that attempts to write to a system file opened under user zero from a nonzero user number. In addition, this error is returned when a program attempts to write to a file password protected in Write mode if the program does not supply the correct password.

The Read-Only Disk error is returned when a program writes to a disk that is in read-only status. A drive can be placed in read-only status explicitly with the BDOS Write Protect Disk function.

The BDOS extended errors are identified by the following error messages:

o Password Error
o File Exists
o ? in Filename

The File Password error is returned when the file password is not supplied, or when it is incorrect. This error is reported only by the banked version of CP/M 3.

The File Exists error is returned by the BDOS Make File and Rename File functions when the BDOS detects a conflict such as a duplicate filename and type.

The ? in Filename error is returned when the BDOS detects a ? in the filename or type field of the passed FCB for the BDOS Rename File, Set File Attributes, Open File, Make File, and Truncate File functions.

The following paragraphs describe the error return code conventions of the BDOS file system functions. Most BDOS file system functions fall into three categories in regard to return codes: they return an Error Code, a Directory Code, or an Error Flag. The error conventions of CP/M 3 are designed to allow programs written for earlier versions of CP/M to run without modification.

The following BDOS functions return an Error Code in register A.

20. Read Sequential
21. Write Sequential
33. Read Random
34. Write Random
40. Write Random w/Zero Fill

The Error Code definitions for register A are shown in Table 2-7.

Table 2-7. Register A BDOS Error Codes

Code	Meaning
00	Function successful
255	Physical error : refer to register H
01	Reading unwritten data or no available directory space (Write Sequential)
02	No available data block
03	Cannot close current extent
04	Seek to unwritten extent
05	No available directory space
06	Random record number out of range
09	Invalid FCB (previous BDOS close call returned an error code and invalidated the FCB)
10	Media Changed (A media change was detected on the FCB's drive after the FCB was opened)

For BDOS read or write functions, the file system also sets register H when the returned Error Code is a value other than zero or 255. In this case, register H contains the number of 128-byte records successfully read or written before the error was encountered. Note that register H can contain only a nonzero value if the calling program has set the BDOS Multi-Sector Count to a value other than one; otherwise register H is set to zero. On successful functions, Error Code = 0, register H is also set to zero. If the Error Code equals 255, register H contains a physical error code (see Table 2-11).

2.3 BDOS File System CP/M 3 Programmer's Guide

The following BDOS functions return a Directory Code in register A:

15. Open File
16. Close File
17. Search For First
18. Search For Next
19. Delete File
22. Make File
23. Rename File
30. Set File Attributes
35. Compute File Size
99. Truncate File

* 100. Set Directory Label
102. Read File Date Stamps and Password Mode
** 103. Write File XFCB

* - This function is supported in the DIRLBL.RSX in the nonbanked version of CP/M 3.
** - This function is supported only in the banked version of CP/M 3.

The Directory Code definitions for register A are shown in Table 2-8.

Table 2-8. BDOS Directory Codes

Code	Meaning
00 - 03:	successful function
255	unsuccessful function

With the exception of the BDOS search functions, all functions in this category return with the directory code set to zero on successful returns. However, for the search functions, a successful Directory Code also identifies the relative starting position of the directory entry in the calling program's current DMA buffer.

If the Set BDOS Error Mode function is used to place the BDOS in return error mode, the following functions return an Error Flag on physical errors:

14. Select Disk
46. Get Disk Free Space
48. Flush Buffers
98. Free Blocks
101. Return Directory Label Data

The Error Flag definition for register A is shown in Table 2-9.

Table 2-9. BDOS Error Flags

Code	Meaning
00	successful function
255	physical error : refer to register H

The BDOS returns nonzero values in register H to identify a physical or extended error if the BDOS Error Mode is in one of the return modes. Except for functions that return a Directory Code, register A equal to 255 indicates that register H identifies the physical or extended error. For functions that return a Directory Code, if register A equals 255, and register H is not equal to zero, register H identifies the physical or extended error. Table 2-10 shows the physical and extended error codes returned in register H.

2.3 BDOS File System CP/M 3 Programmer's Guide

Table 2-10. BDOS Physical and Extended Errors

Code		Meaning
00	-	no error, or not a register H error
01	-	Disk I/O error
02	-	Read-Only Disk
03	-	Read-Only File or File Opened under user zero from another user number or file password protected in write mode and correct password not specified.
04	-	Invalid Drive : drive select error
07	-	Password Error
08	-	File Exists
09	-	? in Filename

The following two functions represent a special case because they return an address in registers H and L.

27. Get Addr(Alloc)
31. Get Addr(Disk Parms)

When the BDOS is in return error mode, and it detects a physical error for these functions, it returns to the calling program with registers A, H, and L all set to 255. Otherwise, they return no error code.

2.4 Page Zero Initialization

Page Zero is the region of memory located from 0000H to 00FFH. This region contains several segments of code and data that are used by transient programs while running under CP/M 3. The code and data areas are shown in Table 2-11 for reference.

Table 2-11. Page Zero Areas

Location From	TO	Contents
0000H	0002H	Contains a jump instruction to the BIOS warm start entry point at BIOS - base + 3. The address at location OOOIH can also be used to make direct BIOS calls to the BIOS console status, console input, console output, and list output primitive functions.
0003H	0004H	(Reserved)
0005H	0007H	Contains a jump instruction to the BDOS, the LOADER, or to the most recently added RSX, and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, and LHLD 0006H places the address field of the jump instruction in the HL register pair. This value, minus one, is the highest address of memory available to the transient program.
0008H	003AH	Reserved interrupt locations for Restarts 1 - 7
003BH	004FH	(Not currently used - reserved)
0050H		Identifies the drive from which the transient program was loaded. A value of one to sixteen identifies drives A through P.
0051H	0052H	Contains the address of the password field of the first command-tail operand in the default DMA buffer beginning at 0080H. The CCP sets this field to zero if no password for the first command-tail operand is specified.
0053H		Contains the length of the password field for the first command-tail operand. The CCP also sets this field to zero if no password for the first command tail is specified.
0054H	0055H	Contains the address of the password field of the second corn-mand-tail operand in the default DMA buffer beginning at 0080H. The CCP sets this field to zero if no password for the second command-tail operand is specified.

Table 2-11. (continued)

Location From TO	Contents
005 6H	Contains the length of the password field for the second corn-mand-tail operand. The CCP also sets this field to zero if no password for the second command tail is specified.
0057H - 005BH	(Not currently used - reserved)
005CH - 007BH	Default File Control Block, FCB, area 1 initialized by the CCP from the first command-tail operand of the command line, if it exists.
006CH - 007BH	Default File Control Block, FCB, area 2 initialized by the CCP from the second command-tail operand of the command line, if it exists. Note: this area overlays the last 16 bytes of default FCB area 1. To use the information in this area, a transient program must copy it to another location before using FCB area 1.
007CH	Current record position of default FCB area 1. This field is used with default FCB area 1 in sequential record processing.
007DH - 007FH	Optional default random record position. This field is an exten-sion of default FCB area 1 used in random record processing.
0080H - OOFFH	Default 128-byte disk buffer. This buffer is also filled with the command tail when the CCP loads a transient program.

The CCP initializes Page Zero prior to initiating a transient program. The fields at 0050H and above are initialized from the command line invoking the transient program. The command line format was described in detail in Section 1.6.2. To summarize, a command line usually takes the form:

<command> <command tail>

where

<command>	=>	<file Spec>
<command tail>	=>	(no command tail)
	=>	<file Spec>
	=>	<file spec><delimiter><file spec>
<file spec>	=>	{d:}filename{.type} {;password}

The CCP initializes the command drive field at 0050H to the drive index, A = 1, P = 16, of the drive from which the transient program was loaded.

The default FCB at 005CH is defined if a command tail is entered. Otherwise, the fields at 005CH, 0068H to 006BH are set to binary zeros, the fields from 005DH to 0067H are set to blanks. The fields at 0051H through 0053H are set if a password is specified for the first <file Spec> of the command tail. If not, these fields are set to zero.

The default FCB at 006CH is defined if a second <file spec> exists in the command tail. Otherwise, the fields at 006CH, 0078H to 007BH are set to binary zeros, the fields from 005DH to 0067H are set to blanks. The fields at 0054H through 0056H are set if a password is specified for the second <file spec> of the command tail. If not, these fields are set to zero.

Transient programs often use the default FCB at 005CH for file operations. This FCB may even be used for random file access because the three bytes starting at 007DH are available for this purpose. However, a transient program must copy the contents of the default FCB at 006CH to another area before using the default FCB at 005CH, because an open operation for the default FCB at 005CH overwrites the FCB data at 006CH.

The default DMA address for transient programs is 0080H. The CCP also initializes this area to contain the command tall of the command line. The first position contains the number of characters in the command line, followed by the command line characters. The character following the last command tail character is set to binary zero. The command line characters are preceded by a leading blank and are translated to ASCII upper-case. Because the 128-byte region beginning at 0080H is the default DMAI, the BDOS file system moves 128-byte records to this area with read operations and accesses 128-byte records from this area with write operations. The transient program must extract the command tail information from this buffer before performing file operations unless it explicitly changes the DMA address with the BDOS Set DMA Address function.

The Page Zero fields of 0051H through OOS6H locate the password fields of the first two file specifications in the command tail if they exist. These fields are provided so that transient programs are not required to parse the command tail for password fields. However, the transient program must save the password, or change the DMA address before performing file operations.

The following example illustrates the initialization of the command line fields of Page Zero. Assuming the following command line is typed at the console:

```
D>A:PROGRAM B:FILE,TYPE ; PASS C ;FILE.TYPE ; PASSWORD
```

A hexadecimal dump of 0050H to 0OA5H would show the Page Zero initialization performed by the CCP.

```
005OH:  01 0D  00 04 9D  00 08 00  00  00 00 00 02 46 49  4C  ...............FIL
006OH:  45 20  20 20  20  54 59 50  00  00 00 00 03 46 49  4C E.... TYP ..... FIL
007OH:  45 20  20 20  20  54 59 50  00  00 00 00 00 00 00  00 E.... TYP ........
0050H:  24 20  42 3A 46  49 4C 45  2E 54 59 50 3B 50 41 53 . .  B:FILE.TYP;PAS
009OH:  53 20  43 3A 46  49 4C 45  2E  54 59 50 3B 50 41 53  S   C:FILE.TYP;PAS
OOAOH:  53 57  4F 52 44  00                                  SWORD.
```

End of Section 2

Section 3

BDOS Function Calls

This section describes each CP/M 3 system function, including the parameters a program must pass when calling the function, and the values the function returns to the program. The functions are arranged numerically for easy reference. You should be familiar with the BDOS calling conventions and other concepts presented in Section., before referencing this section.

BDOS FUNCTION 0: SYSTEM RESET

Entry Parameters:
Register C: 00H

The System Reset function terminates the calling program and returns control to the CCP via a warm start sequence (see Section 1.3.2). Calling this function has the same effect as a 'ump to location 0000H of Page Zero.

Note that the disk subsystem is not reset by System Reset under CP/M 3. The calling program can pass a return code to the CCP by calling Function 108, Get/Set Program Return Code, prior to making a System Reset call or jumping to location 0000H.

BDOS FUNCTION 1: CONSOLE INPUT

Entry Parameters:
Register C: 01H

Returned Value:
Register A: ASCII Character

The Console Input function reads the next character from the logical console, CONIN:, to register A. Graphic characters, along with carriage return, line-feed, and backspace, CTRL-H, are echoed to the console. Tab characters, CTR-L-1, are expanded in columns of 8 characters. CTRL-S, CTRL-Q, and CTRL-P are normally intercepted as described below. All other non-graphic characters are returned in register A but are not echoed to the console.

When the Console Mode is in the default state (see Section 2.2.1), Function 1 intercepts the stop scroll, CTRL-S, start scroll, CTRL-Q, and start/stop printer echo, CTRL-P, characters. Any characters that are typed following a CTRL-S and preceding a CTRL-Q are also intercepted. However, if start/stop scroll has been disabled by the Console Mode, the CTRL-S, CTRL-Q, and CTRL-P characters are not intercepted. Instead, they are returned in register A, but are not echoed to the console.

If printer echo has been invoked, all characters that are echoed to the console are also sent to the list device, LST:.

Function 1 does not return control to the calling program until a non-intercepted character is typed, thus suspending execution if a character is not ready.

BDOS FUNCTION 2: CONSOLE OUTPUT

Entry Parameters:
Registers C: 02H
E: ASCII Character

The Console Output function sends the ASCII character from register E to the logical console device, CONOUT:. When the Console Mode is in the default state (see Section 2.2.1), Function 2 expands tab characters, CTRL-1, in columns of 8 characters, checks for stop scroll, CTRL-S, start scroll, CTRL-Q, and echoes characters to the logical list device, LST:, if printer echo, CTRL-P, has been invoked.

BDOS FUNCTION 3: AUXILIARY INPUT

Entry Parameters:
Register C: 03H

Returned Value:
Register A: ASCII Character

The Auxiliary Input function reads the next character from the logical auxiliary input device, AUXIN:, into register A. Control does not return to the calling program 'I the character is read.
unti

BDOS FUNCTION 4: AUXILIARY OUTPUT

Entry Parameters:
Registers C: 04H
E: ASCII Character

The Auxiliary Output function sends the ASCII character from register E to the logical auxiliary output device, AUXOUT:.

BDOS FUNCTION 5: LIST OUTPUT

Entry Parameters:
Registers C: 05H
E: ASCII Character

The List Output function sends the ASCII character in register E to the logical list device, LST:.

BDOS FUNCTION 6: DIRECT CONSOLE I/0

Entry Parameters:
Registers C: 06H
E: 0FFH (input/status) or
0FEH (status) or
0FDH (input) or
char (output)

Returned Value:
Register A: char or status (no value)

CP/M 3 supports direct I/O to the logical console, CONIN:, for those specialized applications where unadorned console input and output is required. Use Direct Console I/O carefully because it bypasses all the normal control character functions. Programs that perform direct I/O through the BIOS under previous releases of CP/M should be changed to use direct I/O so that they can be fully supported under future releases of MP/M and CP/M.

A program calls Function 6 by passing one of four different values in register E. The values and their meanings are summarized in Table 3-1.

Table 3-1. Function 6 Entry Parameters

Register E value	Meaning
OFFH	Console input/status command returns an input character; if no character is ready, a value of zero is returned.
OFEH	Console status command (On return, register A contains 00 if no character is ready; otherwise it contains FFH.)
OFDH	Console input command, returns an input character; this function will suspend the calling process until a character is ready.
ASCII character	Function 6 assumes that register E contains a valid ASCII character and sends it to the console.

BDOS FUNCTION 7: AUXILIARY INPUT STATUS

Entry Parameters:
Register C: 07H

Returned Value:
Register A: Auxiliary Input Status

The Auxiliary Input Status function returns the value OFFH in register A if a character is ready for input from the logical auxiliary input device, AUXIN:. If no character is ready for input, the value OOH is returned.

BDOS FUNCTION 8: AUXILIARY OUTPUT STATUS

Entry Parameters:
Register C: 08H

Returned Value:
Register A: Auxiliary Output Status

The Auxiliary Output Status function returns the value OFFH in register A if the logical auxiliary output device, AUXOUT:, is ready to accept a character for output. If the device is not ready for output, the value OOH is returned.

BDOS FUNCTION 9: PRINT STRING

Entry Parameters:
Registers C: 09H
DE: String Address

The Print String function sends the character string addressed by register pair DE to the logical console, CONOUT:, until it encounters a delimiter in the string. Usually the delimiter is a dollar sign, $, but it can be changed to any other value by Function 110, Get/Set Output Delimiter. If the Console Mode is in the default state (see Section 2.2.1), Function 9 expands tab characters, CTRL-1, in columns of 8 characters. It also checks for stop scroll, CTRL-S, start scroll, CTRL-Q, and echoes to the logical list device, LST:, if printer echo, CTRL-P, has been invoked.

BDOS FUNCTION 10: READ CONSOLE BUFFER

Entry Parameters:
Registers C: 0AH
DE: Buffer Address

Returned Value:
Console Characters in Buffer

The Read Console Buffer function reads a line of edited console input from the logical console, CONIN:, to a buffer that register pair DE addresses. It terminates input and returns to the calling program when it encounters a return, CTRL-M, or a line feed, CTRL-J, character. Function 10 also discards all input characters after the input buffer is filled. In addition, it outputs a bell character, CTRL-G, to the console when it discards a character to signal the user that the buffer is full. The input buffer addressed by DE has the following format:

Scan and insert illustration here.

where mx is the maximum number of characters which the buffer holds, and nc is the number of characters placed in the buffer. The characters entered by the operator follow the nc value. The value mx must be set prior to making a Function 10 call and may range in value from 1 to 2SS. Setting mx to zero is equivalent to setting mx to one. The value nc is returned to the calling program and may range from zero to mx. If nc < mx, then uninitialized positions follow the last character, denoted by ?? in the figure. Note that a terminating return or line feed character is not placed in the buffer and not included in the count nc.

If register pair DE is set to zero, Function 10 assumes that an initialized input buffer is located at the current DMA address (see Function 26, Set DMA Address). This allows a program to put a string on the screen for the user to edit. To initialize the input buffer, set characters cl through cn to the initial value followed by a binary zero terminator.

When a program calls Function 10 with an initialized buffer, Function 10 operates as if the user had typed in the string. When Function 10 encounters the binary zero terminator, it accepts input from the console. At this point, the user can edit the initialized string or accept it as it is by pressing the RETURN key. However, if the initialized string contains a return, CTRL-M, or a linefeed, CTRL-J, character, Function 10 returns to the calling program without giving the user the opportunity to edit the string.

The level of console editing supported by Function 10 differs for the banked and nonbanked versions of CP/M 3. Refer to the CPIM Plus (CPIM Version 3) Operating System User's Guide for a detailed description of console editing. In the nonbanked version, Function 10 recognizes the edit control characters summarized in Table 3-2.

Table 3-2. Edit Control Characters (Nonbanked CP/M 3)

Character -T	Edit Control Function
rub/del	Removes and echoes the last character; GENCPM can change this function to CTRL-H
CTRL-C	Reboots when at the beginning of line; the Console Mode can disable this function
CTRL-E	Causes physical end of line
CTRL-H	Backspaces one character position; GENCPM can change this function to rub/del
CTRL-J	(Line-feed) terminates input line
CTRL-M	(Return) terminates input line
CTRL-P	Echoes console output to the list device
CTRL-R	Retypes the current line after new line
CTRL-U	Removes current line after new line
CTRL-X	Backspaces to beginning of current line

The banked version of CP/M 3 expands upon the editing provided in the non-banked version. The functionality of the two versions is similar when the cursor is positioned at the end of the line. However, in the banked version, the user can move the cursor anywhere in the current line, insert characters, delete characters, and perform other editing functions. In addition, the banked version saves the previous command line; it can be recalled when the current line is empty. Table 3-3 summarizes the edit control characters supported by Function 10 in the banked version of CP/M 3.

Table 3-3. Edit Control Characters (Banked CP/M 3)

Character	Edit Control Function
rub/del	Removes and echoes the last character if at the end of the line; otherwise deletes the character to the left of the current cursor position; GENCPM can change this function to CTRL-H.
CTRL-A	Moves cursor one character to the left.
CTRL-B	Moves cursor to the beginning of the line when not at the beginning; otherwise moves cursor to the end of the line.
CTRL-C	Reboots when at the beginning of line; the Console Mode can disable this function.
CTRL-E	Causes physical end-of-line; if the cursor is positioned in the middle of a line, the characters at and to the right of the cursor are displayed on the next line.
CTRL-F	Moves cursor one character to the right.
CTRL-G	Deletes the character at the current cursor position when in the middle of the line; has no effect when the cursor is at the end of the line.
CTRL-H	Backspaces one character position when positioned at the end of the line,; otherwise deletes the character to the left of the cursor; GENCPM can change this function to rub/del.

Table 3-3. (continued)

Character	Edit Control Function
CTRL-J	(Line-feed) terminates input; the cursor can be positioned anywhere in the line; the entire input line is accepted; sets the previous line buffer to the input line.
CTRL-K	Deletes all characters to the right of the cursor along with the character at the cursor.
CTRL-M	(Return) terminates input; the cursor can be positioned anywhere in the line; the entire input line is accepted; sets the previous line buffer to the input line.
CTRL-P	Echoes console output to the list device.
CTRL-R	Retypes the characters to the left of the cursor on the new line.
CTRL-U	Updates the previous line buffer to contain the characters to the left of the cursor; deletes current line, and advances to new line.
CTRL-W	Recalls previous line if current line is empty; otherwise moves cursor to end-of-line.
CTRL-X	Deletes all characters to the left of the cursor.

For banked systems, Function 10 uses the console width field defined in the System Control Block. If the console width is exceeded when the cursor is positioned at the end of the line, Function 10 automatically advances to the next line. The beginning of the line can be edited by entering a CTRL-R.

When a character is typed while the cursor is positioned in the middle of the line, the typed character is inserted into the line. Characters at and to the right of the cursor are shifted to the right. If the console width is exceeded, the characters disappear off the right of the screen. However, these characters are not lost. They reappear if characters are deleted out of the line, or if a CTRL-E is typed.

BDOS FUNCTION 11: GET CONSOLE STATUS

Entry Parameters:
Register C: OBH

Returned Value:
Register A: Console Status

The Get Console Status function checks to see if a character has been typed at the logical console, CONIN:. If the Console Mode is in the default state (see Section 2.2.1), Function 11 returns the value 01H in register A when a character is ready. If a character is not ready, it returns a value of 00H.

If the Console Mode is in CTRL-C Only Status mode, Function 11 returns the value 01H in register A only if a CTRL-C has been typed at the console.

BDOS FUNCTION 12: RETURN VERSION NUMBER

Entry Parameters:
Register C: 0CH

Returned Value:
Register HL: Version Number

The Return Version Number function provides information that allows version independent programming. It returns a two-byte value in register pair HL: H contains OOH for CP/M and L contains 31H, the BDOS file system version number. Function 12 is useful for writing applications programs that must run on multiple versions of CP/M and MP/M.

BDOS FUNCTION 13: RESET DISK SYSTEM

Entry Parameters:
Register C: ODH

The Reset Disk System function restores the file system to a reset state where all the disk drives are set to read-write (see Functions 28 and 29), the default disk is set to drive A, and the default DMA address is reset to 0080H. This function can be used, for example, by an application program that requires disk changes during operation. Function 37, Reset Drive, can also be used for this purpose.

BDOS FUNCTION 14: SELECT DISK

Entry Parameters:
Registers C: 0EH

Returned Value: E: Selected Disk

Registers A: Error Flag
H: Physical Error

The Select Disk function designates the disk drive named in register E as the default disk for subsequent BDOS file operations. Register E is set to 0 for drive A, 1 for drive B, and so on through 15 for drive P in a full 16-drive system. In addition, Function 14 logs in the designated drive if it is currently in the reset state. Logging-in a drive activates the drive's directory until the next disk system reset or drive reset operation.

FCBs that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. FCBs with drive code values between 1 and 16, however, gnore the selected default drive and directly reference drives A through P.

Upon return, register A contains a zero if the select operation was successful. If a physical error was encountered, the select function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the 'default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the select function returns to the calling program with register A set to OFFH and register H set to one of the following physical error codes:

01 Disk I/O Error
04 Invalid drive

BDOS FUNCTION 15: OPEN FILE

Entry Parameters:
Registers C: OFH
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical or Extended Error

The Open File function activates the FCB for a file that exists in the disk directory under the currently active user number or user zero. The calling program passes the address of the FCB in register pair DE, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and byte 12 specifying the extent. Usually, byte 12 of the FCB is initialized to zero.

If the file is password protected in Read mode, the correct password must be placed in the first eight bytes of the current DMA, or have been previously established as the default password (see Function 106). If the current record field of the FCB, cr, is set to OFFH, Function 15 returns the byte count of the last record of the file in the cr field. You can set the last record byte count for a file with Function 30, Set File Attributes. Note that the current record field of the FCB, cr, must be zeroed by the calling program before beginning read or write operations if the file is to be accessed sequentially from the first record.

If the current user is non-zero, and the file to be opened does not exist under the current user number, the open function searches user zero for the file. If the file exists under user zero, and has the system attribute, t2', set, the file is opened under user zero. Write operations are not supported for a file that is opened under user zero in this manner.

If the open operation is successful, the user's FCB is activated for read and write operations. The relevant directory information is copied from the matching directory FCB into bytes d0 through dn of the FCB. If the file is opened under user zero when the current user number is not zero, interface attribute f8' is set to one in the user's FCB. In addition, if the referenced file is password protected in Write mode, and the correct password was not passed in the DMA, or did not match the default password, interface attribute f7' is set to one. Write operations are not supported for an activated FCB if interface attribute f7' or f8' is true.

When the open operation is successful, the open function also makes an Access date and time stamp for the opened file when the following conditions are satisfied: the referenced drive has a directory label that requests Access date and time stamping, and the FCB extent number field is zero.

Upon return, the Open File function returns a directory code in register A with the value OOH if the open was successful, or FFH, 255 decimal, if the file was not found. Register H is set to zero in both of these cases. If a physical or extended error was encountered, the Open File function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, the Open File function returns to the calling program with register A set to OFFH, and register H set to one of the following physical or extended error codes:

01 : Disk I/O Error
04 : Invalid drive error
07 : File password error
09 : ? in the FCB filename or filetype field

BDOS FUNCTION 16: CLOSE FILE

Entry Parameters:
Registers C: 10H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical or Extended Error

The Close File function performs the inverse of the Open File function. The calling program passes the address of an FCB in register pair DE. The referenced FCB must have been previously activated by a successful Open or Make function call (see Functions 15 and 22). Interface attribute f5' specifies how the file is to be closed as shown below:

f5' = 0 - Permanent close (default mode)
f5' = 1 - Partial close

A permanent close operation indicates that the program has completed file operations on the file. A partial close operation updates the directory, but indicates that the file is to be maintained in the open state.

If the referenced FCB contains new information because of write operations to the FCB, the close function permanently records the new information in the referenced disk directory. Note that the FCB does not contain new information, and the directory update step is bypassed if only read or update operations have been made to the referenced FCB.

Upon return, the close function returns a directory code in register A with the value 00H if the close was successful,, or FFH, 255 Decimal, if the file was not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the close function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the close function returns to the calling program with register A set to OFFH and register H set to one of the following physical error codes:

01 Disk I/O error
02 Read/only disk
04 Invalid drive error

BDOS FUNCTION 17: SEARCH FOR FIRST

Entry Parameters:
Registers C: 11H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical Error

The Search For First function scans the directory for a match with the FCB addressed by register pair DE. Two types of searches can be performed. For standard searches, the calling program initializes bytes 0 through 12 of the referenced FCB, with byte 0 specifying the drive directory to be searched, bytes 1 through 11 specifying the file or files to be searched for,, and byte 12 specifying the extent. Usually byte 12 is set to zero. An ASCII question mark, 63 decimal, 3F hex, in any of the bytes 1 through 12 matches all entries on the directory in the corresponding position. This facility, called ambiguous reference, can be used to search for multiple files on the directory. When called in the standard mode, the Search function scans for the first file entry in the specified directory that matches the FCB, and belongs to the current user number.

The Search For First function also initializes the Search For Next function. After the Search function has located the first directory entry matching the referenced FCB, the Search For Next function can be called repeatedly to locate all remaining matching entries. In terms of execution sequence, however, the Search For Next call must either follow a Search For First or Search For Next call with no other intervening BDOS disk-related function calls.

If byte 0 of the referenced FCB is set to a question mark, the Search function ignores the remainder of the referenced FCB, and locates the first directory entry residing on the current default drive. All remaining directory entries can be located by making multiple Search For Next calls. This type of search operation is not usually made by application programs, but it does provide complete flexibility to scan all current directory values. Note that this type of search operation must be performed to access a drive's directory label (see Section 2.3.6).

Upon return, the Search function returns a Directory Code in register A with the value 0 to 3 if the search is successful, or OFFH, 255 Decimal, if a matching directory entry is not found. Register H is set to zero in both of these cases. For successful searches, the current DMA is also filled with the directory record containing the matching entry, and the relative starting position is A * 32 (that is, rotate the A register left 5 bits, or ADD A five times). Although it is not usually required for application programs, the directory information can be extracted from the buffer at this position.

If the directory has been initialized for date and time stamping by INITDIR, then an SFCB resides in every fourth directory entry, and successful Directory Codes are restricted to the values 0 to 2. For successful searches, if the matching directory record is an extent zero entry, and if an SFCB resides at offset 96 within the current DMA, contents of (DMA Address + 96) = 21H, the SFCB contains the date and time stamp information, and password mode for the file. This information is located at the relative starting position of 97 + (A * 10) within the current DMA in the following format:

0 - 3 Create or Access Date and Time Stamp Field
4 - 7 Update Date and Time Stamp Field
8 : Password Mode Field

(Refer to Section 2.3.8 for more information on SFCBS.)

If a physical error is encountered, the Search function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the Search function returns to the calling program with register A set to OFFH, and register H set to one of the following physical error codes:

01 Disk I/O error
04 Invalid drive error

BDOS FUNCTION 18: SEARCH FOR NEXT

Entry Parameters:
Register C: 12H

Returned Value:
Registers A: Directory Code
H: Physical Error

The Search For Next function is identical to the Search For First function, except that the directory scan continues from the last entry that was matched. Function 18 returns a Directory code in register A, analogous to Function 17.

Note: in execution sequence, a Function 18 call must follow either a Function 17 or another Function 18 call with no other intervening BDOS disk-related function calls.

BDOS FUNCTION 19: DELETE FILE

Entry Parameters:
Registers C: 13H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Extended or Physical Error

The Delete File function removes files or XFCBs that match the FCB addressed in register pair DE. The filename and filetype can contain ambiguous references, that is, question marks 'in bytes fl through t3, but the dr byte cannot be ambiguous, as it can in the Search and Search Next functions. Interface attribute f5' specifies the type of delete operation that is performed.

f5' = 0 - Standard Delete (default mode)
f5' = 1 - Delete only XFCBs

If any of the files that the referenced FCB specify are password protected, the correct password must be placed in the first eight bytes of the current DMA buffer, or have been previously established as the default password (see Function 106).

For standard delete operations, the Delete function removes all directory entries belonging to files that match the referenced FCB. All disk directory and data space owned by the deleted files is returned to free space, and becomes available for allocation to other files. Directory XFCBs that were owned by the deleted files are also removed from the directory. If interface attribute f5' of the FCB is set to 1, Function 19 deletes only the directory XFCBs that match the referenced FCB.

Note: if any of the files that match the input FCB specification fall the password check, or are Read-Only, then the Delete function does not delete any files or XFCBS. This applies to both types of delete operations.

In nonbanked systems, file passwords and XFCBs are not supported. Thus, if the Delete function is called with interface attribute f5' set to true, the Delete function performs no action but returns with register A set to zero.

Upon return, the Delete function returns a Directory Code in register A with the value 0 if the delete is successful, or OFFH, 255 Decimal, if no file that matches the referenced FCB is found. Register H is set to zero in both of these cases. If a physical, or extended error is encountered, the Delete function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the Delete function returns to the calling program with register A set to OFFH and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
03 : Read-Only file
04 : Invalid drive error
07 : File password error

BDOS FUNCTION 20: READ SEQUENTIAL

Entry Parameters:
Registers C: 14H
DE: FCB Address

Returned Value:
Registers A: Error Code
H: Physical Error

The Read Sequential function reads the next 1 to 128 128-byte records from a file into memory beginning at the current DMA address. The BDOS Multi-Sector Count (see Function 44) determines the number of records to be read. The default is one record. The FCB addressed by register pair DE must have been previously activated by an Open or Make function call.

Function 20 reads each record from byte cr of the extent, then automatically increments the cr field to the next record position. If the cr field overflows, then the function automatically opens the next logical extent and resets the cr field to 0 in preparation for the next read operation. The calling program must set the cr field to 0 following the Open call if the intent is to read sequentially from the beginning of the file.

Upon return, the Read Sequential function sets register A to zero if the read operation is successful. Otherwise, register A contains an error code identifying the error as shown below:

01 Reading unwritten data (end-of-file)
09 Invalid FCB
10 Media change occurred
255 Physical Error; refer to register H

Error Code 01 is returned if no data exists at the next record position of the file. Usually, the no data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block that has not been previously written, or an extent which has not been created. These situations are usually restricted to files created or appended with the BDOS random write functions (see Functions 34 and 40).

Error Code 09 is returned if the FCB is invalidated by a previous BDOS close call that returns an error.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS Open, or Make Call.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is Return Error mode, or Return and Display Error mode (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

01 Disk I/O error
04 Invalid drive error

On all error returns except for physical error returns, A = 255, Function 20 sets register H to the number of records successfully read before the error is encountered. This value can range from 0 to 127 depending on the current BDOS Multi-Sector Count. It is always set to zero when the Multi-Sector Count is equal to one.

BDOS FUNCTION 21: WRITE SEQUENTIAL

Entry Parameters:
Registers C: 15H
DE: FCB Address

Returned Value:
Registers A: Error Code
H: Physical Error

The Write Sequential function writes 1 to 128 128-byte data records, beginning at the current DMA address into the file named by the FCB addressed in register pair DE. The BDOS Multi-Sector Count (see Function 44) determines the number of 128 byte records that are written. The default is one record. The referenced FCB must have been previously activated by a BDOS Open or Make function call.

Function 21 places the record into the file at the position indicated by the cr byte of the FCB, and then automatically increments the cr byte to the next record position. If the cr field overflows, the function automatically opens, or creates the next logical extent, and resets the cr field to 0 in preparation for the next write operation. If Function 21 is used to write to an existing file, then the newly written records overlay those already existing in the file. The calling program must set the cr field to 0 following an Open or Make call if the intent is to write sequentially from the beginning of the file.

Function 21 makes an Update date and time for the file if the following conditions are satisfied: the referenced drive has a directory label that requests date and time stamping, and the file has not already been stamped for update by a previous Make or Write function call.

Upon return, the Write Sequential function sets register A to zero if the write operation is successful. Otherwise, register A contains an error code identifying the error as shown below:

01 No available directory space
02 No available data block
09 Invalid FCB
10 Media change occurred
2S5 Physical Error : refer to register H

Error Code 01 is returned when the write function attempts to create a new extent that requires a new directory entry, and no available directory entries exist on the selected disk drive.

Error Code 02 is returned when the write command attempts to allocate a new data block to the file, and no unallocated data blocks exist on the selected disk drive.

Error Code 09 is returned if the FCB is invalidated by a previous BDOS close call that returns an error.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS Open or Make call.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is Return Error mode, or Return and Display Error mode (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

01	Disk I/O error
02	Read-Only disk
03	Read-Only file or File open from user 0 when the current user number is non-zero or File password protected in Write mode
04	Invalid drive error

On all error returns, except for physical error returns, A = 255, Function 21 sets register H to the number of records successfully written before the error was encountered. This value can range from 0 to 127 depending on the current BDOS Multi-Sector Count. It is always set to zero when the Multi-Sector Count is set to one.

BDOS FUNCTION 22: MAKE FILE

Entry Parameters:
Registers C: 16H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical or Extended Error

The Make File function creates a new directory entry for a file under the current user number. It also creates an XFCB for the file if the referenced drive has a directory label that enables password protection on the drive, and the calling program assigns a password to the file.

The calling program passes the address of the FCB in register pair DE, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filctype, and byte 12 set to the extent number. Usually, byte 12 is set to zero. Byte 32 of the FCB, the cr field, must be initialized to zero, before or after the Make call, if the intent is to write sequentially from the beginning of the file.

Interface attribute f6' specifies whether a password is to be assigned to the created file.

f6' = 0 - Do not assign password (default)
f6' = 1 - Assign password to created file

When attribute f6' is set to 1, the calling program must place the password in the first 8 bytes of the current DMA buffer, and set byte 9 of the DMA buffer to the password mode (see Function 102). Note that the Make function only interrogates interface attribute f6' if passwords are activated on the referenced drive. In non-banked systems, file passwords are not supported, and attribute f6' is never interrogated.

The Make function returns with an error if the referenced FCB names a file that currently exists in the directory under the current user number.

If the Make function is successful, it activates the referenced FCB for file operations by opening the FCB, and initializes both the directory entry and the referenced FCB to an empty file. It also initializes all file attributes to zero. In addition, Function 22 makes a Creation date and time stamp for the file if the following conditions are satisfied: the referenced drive has a directory label that requests Creation date and time stamping and the FCB extent number field is equal to zero. Function 22 also makes an Update stamp if the directory label requests update stamping and the FCB extent field is equal to zero.

If the referenced drive contains a directory label that enables password protection, and if interface attribute f6' has been set to 1, the Make function creates an XFCB for the file. In addition, Function 22 also assigns the password, and password mode placed in the first nine bytes of the DMA, to the XFCB.

Upon return, the Make function returns a directory code in register A with the value 0 if the make operation is successful, or OFFH, 25S decimal, if no directory space is available. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the Make function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the Make function returns to the calling program with register A set to OFFH, and register H set to one of the following physical or extended error codes:

01 : Disk I/O error
02 : Read-Only disk
04 : Invalid drive error
08 : File already exists
09 : ? in filename or filetype field

BDOS FUNCTION 23: RENAME FILE

Entry Parameters:
Registers C: 17H
DE: FCB Address

Returned Value-.
Registers A: Directory Code
H: Physical or Extended Error

The Rename function uses the FCB, addressed by register pair DE, to change all directory entries of the file specified by the filename in the first 16 bytes of the FCB to the filename in the second 16 bytes. If the file specified by the first filename is password protected, the correct password must be placed in the first eight bytes of the current DMA buffer, or have been previously established as the default password (see Function 106). The calling program must also ensure that the filenames specified in the FCB are valid and unambiguous, and that the new filename does not already exist on the drive. Function 23 uses the dr code at byte 0 of the FCB to select the drive. The drive code at byte 16 of the FCB is ignored.

Upon return, the Rename function returns a Directory Code in register A with the value 0 if the rename is successful, or 0FFH, 255 Decimal, if the file named by the first filename in the FCB is not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the Rename function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, the Rename function returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

01	Disk I/O error
02	Read-Only disk
03	Read-Only file
04	Invalid drive error
07	File password error
08	File already exists
09	? in filename or filetype field

BDOS FUNCTION 24: RETURN LOGIN VECTOR

Entry Parameters:
Register C: 18H

Returned Value:
Register HL: Login Vector

Function 24 returns the login vector in register pair HL. The login vector is a 16-bit value with the least significant bit of L corresponding to drive A, and the high-order bit of H corresponding to the 16th drive, labelled P. A 0 bit indicates that the drive is not on-line, while a 1 bit indicates the drive is active. A drive is made active by either an explicit BDOS Select Disk call, number 14, or an implicit selection when a BDOS file operation specifies a non-zero dr byte in the FCB. Function 24 maintains compatibilty with earlier releases since registers A and L contain the same values upon return.

BDOS FUNCTION 25: RETURN CURRENT DISK

Entry Parameters:
Register C: 19H

Returned Value:
Register A: Current Disk

Function 25 returns the currently selected default disk number in register A. The disk numbers range from 0 through 15 corresponding to drives A through P.

BDOS FUNCTION 26: SET DMA ADDRESS

Entry Parameters:
Registers C: 1AH
DE: DMA Address

DMA is an acronym for Direct Memory Address, which is often used in connection with disk controllers that directly access the memory of the computer to transfer data to and from the disk subsystem. Under CP/M 3, the current DMA is usually defined as the buffer in memory where a record resides before a disk write, and after a disk read operation. If the BDOS Multi-Sector Count is equal to one (see Function 44), the size of the buffer is 128 bytes. However, if the BDOS Multi-Sector Count is greater than one, the size of the buffer must equal N * 128, where N equals the Multi-Sector Count.

Some BDOS functions also use the current DMA to pass parameters, and to return values. For example, BDOS functions that check and assign file passwords require that the password be placed in the current DMA. As another example, Function 46, Get Disk Free Space, returns its results in the first 3 bytes of the current DMA. When the current DMA is used in this context, the size of the buffer in memory is determined by the specific requirements of the called function.

When a transient program is initiated by the CCP, its DMA address is set to 0080H. The BDOS Reset Disk System function, Function 13, also sets the DMA address to 0080H. The Set DMA function can change this default value to another memory address. The DMA address is set to the value passed in the register pair DE. The DMA address remains at this value until it is changed by another Set DMA Address, or Reset Disk System call.

BDOS FUNCTION 27: GET ADDR(ALLOC)

Entry Parameters:
Register C: 1BH

Returned Value:
Register HL: ALLOC Address

CP/M 3 maintains an allocation vector in main memory for each active disk drive. Some programs use the information provided by the allocation vector to determine the amount of free data space on a drive. Note, however, that the allocation information might be inaccurate if the drive has been marked Read-Only.

Function 27 returns in register pair HL, the base address of the allocation vector for the currently selected drive. If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 4S), Function 27 returns the value OFFFFH in the register pair HL.

In banked CP/M 3 systems, the allocation vector can be placed in bank zero. In this case, a transient program cannot access the allocation vector. However, the BDOS function, Get Disk Free Space (Function 46), can be used to directly return the number of free 128-byte records on a drive. The CP/M 3 utilities that display a drive's free space, DIR and SHOW, use Function 46 for that purpose.

BDOS FUNCTION 28: WRITE PROTECT DISK

Entry Parameters:
Register C: 1CH

The Write Protect Disk function provides temporary write protection for the currently selected disk by marking the drive as Read-Only, No program can write to a disk that is in the Read-Only state. A drive reset operation must be performed for a Read-Only drive to restore it to the Read-Write state (see Functions 13 and 37).

BDOS FUNCTION 29: GET READ-ONLY VECTOR

Entry Parameters:
Register C: lDH

Returned Value:
Register HL: R/O Vector Value

Function 29 returns a bit vector in register pair HL that indicates which drives have the temporary Read-Only bit set. The Read-Only bit can be set only by a BDOS Write Protect Disk call.

The format of the bit vector is analogous to that of the login vector returned by Function 24. The least significant bit corresponds to drive A, while the most significant bit corresponds to drive P.

BDOS FUNCTION 30: SET FILE ATTRIBUTES

Entry Parameters:
Registers C: 1EH
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical or Extended error

By calling the Set File Attributes function, a program can modify a file's attributes and set its last record byte count. Other BDOS functions can be called to interrogate these file parameters, but only Function 30 can change them. The file attributes that can be set or reset by Function 30 are fl' through f4', Read-Only, tl', System, t2', and Archive, t3'. The register pair DE addresses an FCB containing a filename with the appropriate attributes set or reset. The calling program must ensure that it does not specify an ambiguous filename. In addition, if the specified file is password to-tected, the correct password must be placed in the first eight bytes of the current DMA buffer or have been previously established as the default password (see Function 106).

Interface attribute f6' specifies whether the last record byte count of the specified file is to be set:

f6' = 0 - Do not set byte count (default mode)
f6' = 1 - Set byte count

If interface attribute f6' is set, the calling program must set the cr field of the referenced FCB to the byte count value. A program can access a file's byte count value with the BDOS Open, Search, or Search Next functions.

Function 30 searches the referenced directory for entries belonging to the current user number that matches the FCB specified name and type fields. The function then updates the directory to contain the selected indicators, and if interface attribute f6' is set, the specified byte count value. Note that the last record byte count is maintained in byte 13 of a file's directory FCBS.

File attributes tl', t2', and t3' are defined by CP/M 3. (They are described in Section 2.3.4.) Attributes fl' through f4' are not presently used, but can be useful for application programs, because they are not involved in the matching program used by the BDOS during Open File and Close File operations. Indicators f5' through f8' are reserved for use as interface attributes.

Upon return, Function 30 returns a Directory Code in register A with the value 0 if the function is successful, or OFFH, 255 Decimal, if the file specified by the referenced FCB is not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered', the Set File Attributes function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the program is terminated. Otherwise, Function 30 returns to the calling program with reg'ls-Ler A set to OFFH, and register H set to one of the following physical or extended error codes:

01 Disk I/O error
02 Read-Only disk
04 Select error
07 File password error
09 ? in filename or filetype field

BDOS FUNCTION 31: GET ADDR(DPB PARMS)

Entry Parameters:
Register C: 1FH

Returned Value:
Register HL: DPB Address

Function 31 returns in register pair HL the address of the BIOS-resident Disk Parameter Block, DPB, for the currently selected drive. (Refer to the CP/M Plus (CP/M Version 3) Operating System System Guide for the format of the DPB). The calling program can use this address to extract the disk parameter values.

If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), Function 31 returns the value OFFFFH in the register pair HL.

BDOS FUNCTION 32: SET/GET USER CODE

Entry Parameters:
Registers C: 20H

Returned Value: E: OFFH (get) or User Code (set)

Register A: Current Code or
(no value)

A program can change, or interrogate the currently active user number by calling Function 32. If register E = OFFH, then the value of the current user number is returned in register A, where the value is in the range of 0 to 15. If register E is not OFFH, then the current user number is changed to the value of E, modulo 16.

BDOS FUNCTION 33: READ RANDOM

Entry Parameters:
Registers C: 21H
DE: FCB Address

Returned Value:
Registers A: Error Code
H: Physical Error

The Read Random function is similar to the Read Sequential function except that the read operation takes place at a particular random record number, selected by the 24-bit value constructed from the three byte, rO, rl, r2, field beginning at position 33 of the FCB. Note that the sequence of 24 bits is stored with the least significant byte first, rO, the middle byte next, rl, and the high byte last, r2. The random record number can range from 0 to 262,143. This corresponds to a maximum value of 3 in byte r2.

To read a file with Function 33, the calling program must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. The base extent may or may not contain any allocated data. Function 33 reads the record specified by the random record field into the current DMA address. The function automatically sets the logical extent and current record values, but unlike the Read Sequential function, it does not advance the current record number. Thus, a subsequent Read Random call rereads the same record. After a random read operation, a file can be accessed sequentially, starting from the current randomly accessed position. However, the last randomly accessed record is reread or rewritten when switching from random to sequential mode.

If the BDOS Multi-Sector Count is greater than one (see Function 44), the Read Random function reads multiple consecutive records into memory beginning at the current DMA. The rO, rl, and r2 field of the FCB is automatically incremented to read each record. However, the FCBs random record number is restored to the first record's value upon return to the calling program.

Upon return, the Read Random function sets register A to zero if the read operation was successful. Otherwise, register A contains one of the following error codes:

01 Reading unwritten data (end-of-file)
03 Cannot close current extent
04 Seek to unwritten extent
06 Random record number out of range
10 Media change occurred
255 Physical Error : refer to register H

Error Code 01 is returned if no data exists at the next record position of the file. Usually, the no data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block that has not been previously written.

Error Code 03 is returned when the Read Random function cannot close the current extent prior to moving to a new extent.

Error Code 04 is returned when a read random operation accesses an extent that has not been created.

Error Code 06 is returned when byte 35, r2, of the referenced FCB is greater than 3.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS Open or Make Call.

Error Code 255 is returned if a physical error is encountered, and the BDOS error mode is one of the return modes (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

01 Disk I/O error
04 Invalid drive error

On all error returns except for physical errors, A = 255, the Read Random function sets register H to the number of records successfully read before the error is encountered. This value can range from 0 to 127 depending on the current BDOS Multi-Sector Count. It is always set to zero when the Multi-Sector Count is equal to one.

BDOS FUNCTION 34: WRITE RANDOM

Entry Parameters:
Registers C: 22H
DE: FCB Address

Returned Value:
Registers A: Error Code
H: Physical Error

The Write Random function is analogous to the Read Random function, except that data is written to the disk from the current DMA address. If the disk extent or data block where the data is to be written is not already allocated, the BDOS automatically performs the allocation before the write operation continues.

To write to a file using the Write Random function, the calling program must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. If the file is empty, the calling program must create the base extent with the Make File function before calling Function 34. The base extent might or might not contain any allocated data, but it does record the file in the directory, so that the file can be displayed by the DIR utility.

The Write Random function sets the logical extent and current record positions to correspond with the random record being written, but does not change the random record number. Thus, sequential read or write operations can follow a random write, with the current record being reread or rewritten as the calling program switches from random to sequential mode.

Function 34 makes an Update date and time stamp for the file if the following conditions are satisfied: the referenced drive has a directory label that requests Update date and time stamping if the file has not already been stamped for update by a previous BDOS Make or Write call.

If the BDOS Multi-Sector Count is greater than one (see Function 44), the Write Random function reads multiple consecutive records into memory beginning at the current DMA. The rO, rl, and r2 field of the FCB is automatically incremented to write each record. However, the FCB's random record number is restored to the first record's value when it returns to the calling program. Upon return, the Write Random function sets register A to zero if the write operation is successful. Otherwise, register A contains one of the following error codes:

02 No available data block
03 Cannot Close current extent
05 No available directory space
06 Random record number out of range
10 Media change occurred
255 Physical Error : refer to register H

Error Code 02 is returned when the write command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

Error Code 03 is returned when the Write Random function cannot close the current extent prior to moving to a new extent.

Error Code 05 is returned when the write function attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.

Error Code 06 is returned when byte 35, r2, of the referenced FCB is greater than 3.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS Open or Make Call.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is one of the return modes (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, it is identified by register H as shown below:

01 Disk I/O error
02 Read-Only disk
03 Read-Only file or
File open from user 0 when the current user number is nonzero or
File password protected in Write mode
04 Invalid drive error

On all error returns, except for physical errors, A = 255, the Write Random function sets register H to the number of records successfully written before the error is encountered. This value can range from 0 to 127 depending on the current BDOS Multi-Sector Count. It is always set to zero when the Multi-Sector Count is equal to one.

BDOS FUNCTION 35: COMPUTE FILE SIZE

Entry Parameters:
Registers C: 23H
DE: FCB Address

Returned Value:
Registers A: Error Flag
H: Physical or Extended error

Random Record Field Set

The Compute File Size function determines the virtual file size, which is, in effect, the address of the record immediately following the end of the file. The virtual size of a file corresponds to the physical size if the file is written sequentially. If the file is written in random mode, gaps might exist in the allocation, and the file might contain fewer records than the indicated size. For example, if a single record with record number 262,143, the CP/M 3 maximum is written to a file using the Write Random function, then the virtual size of the file is 262,144 records even though only 1 data block 'is actually allocated.

To compute file size, the calling program passes in register pair DE the address of an FCB in random mode format, bytes rO, rl and r2 present. Note that the FCB must contain an unambiguous filename and filetype. Function 35 sets the random record field of the FCB to the random record number + 1 of the last record in the file. If the r2 byte is set to 04, then the file contains the maximum record count 262,144.

A program can append data to the end of an existing file by calling Function 35 to set the random record position to the end of file, and then performing a sequence of random writes starting at the preset record address.

Note: the BDOS does not require that the file be open to use Function 35. However, if the file has been written to, it must be closed before calling Function 35. Otherwise, an incorrect file size might be returned.

Upon return, Function 35 returns a zero in register A if the file specified by the referenced FCB is found, or an OFFH in register A if the file is not found. Register H is set to zero in both of these cases. If a physical error is encountered, Function 35 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, Function 35 returns to the calling program with register A set to OFFH, and register H set to one of the following physical errors:

01 Disk I/O error
04 Invalid drive error

BDOS FUNCTION 36: SET RANDOM RECORD

Entry Parameters:
Registers C: 24H
DE: FCB Address

Returned Value: Random Record Field Set

The Set Random Record function returns the random record number of the next record to be accessed from a file that has been read or written sequentially to a particular point. This value is returned in the random record field, bytes rO, rl, and r2, of the FCB addressed by the register pair DE. Function 36 can be useful in two ways,

First, it is often necessary to initially read and scan a sequential file to extract the positions of various key fields. As each key is encountered, Function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record number minus one is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move directly to a particular record by performing a random read using the corresponding random record number that you saved earlier. The scheme is easily generalized when variable record lengths are involved, because the program need only store the buffer-relative byte position along with the key and record number to find the exact starting position of the keyed data at a later time.

A second use of Function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, then Function 36 is called to set the record number, and subsequent random read and write operations continue from the next record in the file.

BDOS FUNCTION 37: RESET DRIVE

Entry Parameters:
Registers C: 25H
DE: Drive Vector

Returned Value:
Register A: 00H

The Reset Drive function programmatically restores specified drives to the reset state. A reset drive is not logged-in and is in Read-Write status. The passed parameter in register pair DE is a 16-bit vector of drives to be reset, where the least significant bit corresponds to the first drive A, and the high-order bit corresponds to the sixteenth drive, labelled P. Bit values of 1 indicate that the specified drive is to be reset.

BDOS FUNCTION 38: ACCESS DRIVE

Entry Parameters:
Register C: 26H

This is an MP/M function that is not supported under CP/M 3. If called, the file system returns a zero In register A indicating that the access request is successful.

BDOS FUNCTION 39: FREE DRIVE

Entry Parameters:
Register C: 27H

This is an MP/M function that is not supported under CP/M 3. If called, the file system returns a zero In register A indicating that the free request is successful.

BDOS FUNCTION 40: WRITE RANDOM WITH ZERO FILL

Entry Parameters:
Registers C: 28H
DE: FCB address

Returned Value:
Registers A: Error Code
H: Physical Error

The Write Random With Zero Fill function is identical to the Write Random function (Function 34) with the exception that a previously unallocated data block is filled with zeros before the record is written. If this function has been used to create a file, records accessed by a read random operation that contain all zeros identify unwritten random record numbers. Unwritten random records in allocated data blocks of files created using the Write Random function (Function 34) contain uninitialized data.

BDOS FUNCTION 41: TEST AND WRITE RECORD

Entry Parameters:
Registers C: 29H
DE: FCB Address

Returned Value:
Registers A: Error Code
H: Physical Error

The Test and Write Record function is an MP/M 11" function that is not supported under CP/M 3. If called, Function 41 returns with register A set to OFFH and register H set to zero.

BDOS FUNCTION 42: LOCK RECORD

Entry Parameters:
Registers C: 2AH
DE: FCB Address

Returned Value:
Register A: 00H

The Lock Record function is an MP/M II function that is supported under CP/M 3 only to provide compatibility between CP/M 3 and MP/M. It is intended for use in situations where more than one running program has Read-Write access to a common file. Because CP/M 3 is a single-user operating system in which only one program can run at a time, this situation cannot occur. Thus, under CP/M 3, Function 42 performs no action except to return the value 00H in register A indicating that the record lock operation is successful.

BDOS FUNCTION 43: UNLOCK RECORD

Entry Parameters:
Registers C: 2BH
DE: FCB Address

Returned Value:
Register A: 00H

The Unlock Record function is an MP/M II function that is supported under CP/M 3 only to provide compatibility between CP/M 3 and MP/M. It is intended for use in situations where more than one running program has Read-Write access to a common file. Because CP/M 3 is a single-user operating system in which only one program can run at a time, this situation cannot occur. Thus, under CP/M 3, Function 43 performs no action except to return the value 00H in register A indicating that the record unlock operation is successful.

BDOS FUNCTION 44: SET MULTI-SECTOR COUNT

Entry Parameters:
Registers C: 2CH
E: Number of Sectors

Returned Value:
Register A: Return Code

The Set Multi-Sector Count function provides logical record blocking under CP/M 3. It enables a program to read and write from 1 to 128 records of 128 bytes at a time during subsequent BDOS Read and Write functions.

Function 44 sets the Multi-Sector Count value for the calling program to the value passed in register E. Once set, the specified Multi-Sector Count remains in effect until the calling program makes another Set Multi-Sector Count function call and changes the value. Note that the CCP sets the Multi-Sector Count to one when it initiates a transient program.

The Multi-Sector Count affects BDOS error reporting for the BDOS Read and Write functions. If an error interrupts these functions when the Multi-Sector is greater than one, they return the number of records successfully read or written in register H for all errors except for physical errors (A = 255).

Upon return, register A is set to zero if the specified value is in the range of 1 to 128. Otherwise, register A is set to 0FFH.

BDOS FUNCTION 45: SET BDOS ERROR MODE

Entry Parameters:
Registers C: 2DH
E: BDOS Error Mode

Returned Value: None

Function 45 sets the BDOS error mode for the calling program to the mode specified in register E. If register E is set to OFFH, 255 decimal, the error mode is set to Return Error mode. If register E is set to OFEH, 254 decimal, the error mode is set to Return and Display mode. If register E is set to any other value, the error mode is set to the default mode.

The SET BDOS Error Mode function determines how physical and extended errors (see Section 2.2.13) are handled for a program. The Error Mode can exist in three modes: the default mode, Return Error mode, and Return and Display Error mode. In the default mode, the BDOS displays a system message at the console that identifies the error and terminates the calling program. In the return modes, the BDOS sets register A to OFFH, 255 decimal, places an error code that identifies the physical or extended error in register H and returns to the calling program. In Return and Display mode, the BDOS displays the system message before returning to the calling program. No system messages are displayed, however, when the BDOS is in Return Error mode.

BDOS FUNCTION 46: GET DISK FREE SPACE

Entry Parameters:
Registers C: 2EH
E: Drive

Returned Value: First 3 bytes of current DMA buffer
Registers A: Error Flag
H: Physical Error

The Get Disk Free Space function determines the number of free sectors, 128 byte records, on the specified drive. The calling program passes the drive number in register E, with 0 for drive A, 1 for B, and so on, through 15 for drive P in a full 16-drive system. Function 46 returns a binary number in the first 3 bytes of the current DMA buffer. This number is returned in the following format:

fso fsl fs2

Disk Free Space Field Format

fso = low byte
fsl = middle byte
fs2 = high byte

Note that the returned free space value might be inaccurate if the drive has been marked Read-Only.

Upon return, register A is set to zero if the function is successful. However, if the BDOS Error Mode is one of the return modes (see Function 45), and a physical error is encountered, register A is set to OFFH, 255 decimal, and register H is set to one of the following values:

01 - Disk I/O error
04 - Invalid drive error

BDOS FUNCTION 47: CHAIN TO PROGRAM

Entry Parameters:
Registers C: 2FH
E: Chain Flag

The Chain To Program function provides a means of chaining from one program to the next without operator intervention. The calling program must place a command line terminated by a null byte, OOH, in the default DMA buffer. If register E is set to 0FFH, the CCP initializes the default drive and user number to the current program values when it passes control to the specified transient program. Otherwise, these parameters are set to the default CCP values. Note that Function 108, Get/Set Program Return Code, can be used to pass a two byte value to the chained program.

Function 47 does not return any values to the calling program and any encountered errors are handled by the CCP.

BDOS FUNCTION 48: FLUSH BUFFERS

Entry Parameters:
Registers C: 30H

Returned Value: E: Purge Flag

Registers A: Error Flag
H: Physical Error

The Flush Buffers function forces the write of any write-pending records contained in internal blocking/deblocking buffers. If register E is set to OFFH, this function also purges all active data buffers. Programs that provide write with read verify support need to purge internal buffers to ensure that verifying reads actually access the disk instead of returning data that is resident in internal data buffers. The CP/M 3 PIP utility is an example of such a program.

Upon return, register A is set to zero if the flush operation is successful. If a physical error is encountered, the Flush Buffers function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode,, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the Flush Buffers function returns to the calling program with register A set to OFFH and register H set to the following physical error code:

0 1 Disk I/O error
02 Read/only disk
04 Invalid drive error

BDOS FUNCTION 49: GET / SET SYSTEM CONTROLBLOCK

Entry Parameters:
Registers C: 31H
DE: SCB PB Address

Returned Value:
Registers A: Returned Byte
HL: Returned Word

Function 49 allows access to parameters located in the CP/M 3 System Control Block (SCB). The SCB is a 100-byte data structure residing within the BDOS that contains flags and data used by the BDOS, CCP and other system components. Note that Function 49 is a CP/M 3 specific function. Programs intended for both MP/M 11 and CP/M 3 should either avoid the use of this function or isolate calls to this function in CP/M 3 version-dependent sections.

To use Function 49, the calling program passes the address of a data structure called the SCB parameter block in register pair DE. This data structure identifies the byte or word of the SCB to be updated or returned. The SCB parameter block is defined as:

```
SCBPB:      DB OFFSET     ; Offset within SCB
            DB SET        ; OFFH if settin!i a byte
                          ; OFEH if setting a word
                          ; 001H - OFDH are reserved
                          ; OOOH if a get operation
            DW VALUE      ; Byte or word value to be set
```

The OFFSET parameter identifies the offset of the field within the SCB to be updated or accessed. The SET parameter determines whether Function 49 is to set a byte or word value in the SCB or if it is to return a byte from the SCB. The VALUE parameter is used only in set calls. In addition, only the first byte of VALUE is referenced in set byte calls.

Use caution when you set SCB fields. Some of these parameters reflect the current state of the operating system. If they are set to invalid values, software errors can result. In general, do not use Function 49 to set a system parameter if another BDOS function can achieve the same result. For example, Function 49 can be called to update the Current DMA Address field within the SCB. This is not equivalent to making a Function 26, Set DMA Address call, and updating the SCB Current DMA field in this way would result in system errors. However, you can use Function 49 to return the Current DMA address. The System Control Block is summarized in the following table. Each of these fields is documented in detail in Appendix A.

Table 3-4. System Control Block

Offset -7	Description
00 - 04	Reserved For System Use
05	BDOS version number
06 - 09	User Flags
OA - OF	Reserved For System Use
10 - 11	Program Error return code
12 - 19	Reserved For System Use
1A	Console Width (columns)
1B	Console Column Position
1C	Console Page Length
1D - 21	Reserved For System Use
22 - 23	CONIN Redirection flag
24 - 25	CONOUT Redirection flag
26 - 27	AUXIN Redirection flag
28 - 29	AUXOUT Redirection flag
2A - 2B	LSTOUT Redirection flag
2C	Page Mode
2D	Reserved For System Use
2E	CTRL-H Active
2F	Rubout Active
30 - 32	Reserved For System Use
33 - 34	Console Mode
35 - 36	Reserved For System Use
37	Output Delimiter
38	List Output Flag
39 - 3B	Reserved For System Use

Table 3-4. (continued)

Offset	Description
3C-3D	Current DMA Address
3E	Current Disk
3F - 43	Reserved For System Use
44	Current User Number
45 - 49	Reserved For System Use
4A	BDOS Multi-Sector Count
4B	BDOS Error Mode
4C - 4F	Drive Search Chain (DISKS A:,E:,F:)
50	Temporary File Drive
51	Error Disk
52 - 56	Reserved For System Use
57	BDOS flags
58 - 5C	Date Stamp
5D - 5E	Common Memory Base Address
5F - 63	Reserved For System Use

If Function 49 is called with the OFFSET parameter of the SCB parameter block greater than 63H, the function performs no action but returns with registers A and HL set to zero.

BDOS FUNCTION 50: DIRECT BIOS CALLS

Entry Parameters:
Registers C: 32H
DE: BIOS PB Address

Returned Value: BIOS RETURN

Function 50 provides a direct BIOS call through the BDOS to the BIOS. The calling program passes the address of a data structure called the BIOS Parameter Block (BIOSPB) in register pair DE. The BIOSPB contains the BIOS function number and register contents as shown below:

```
BIOSPB:  db FUNC       ; BIOS function no.
         db  AREG      ; A register contents
         dw  BCREG     ; BC register contents
         dw  DEREG     ; DE register contents
         dw  HLREG     ; HL register contents
```

System Reset (Function 0) is equivalent to Function 50 with a BIOS function number of 1.

Note that the register pair BIOSPB fields (BCREG, DEREG, HLREG) arc defined in low byte, high byte order. For example, in the BCREG field, the first byte contains the C register value, the second byte contains the B register value.

Under CP/M 3, direct BIOS calls via the BIOS jump vector are only supported for the BIOS Console I/O and List functions. You must use Function 50 to call any other BIOS functions. In addition, Function 50 intercepts BIOS Function 27 (Select Memory) calls and returns with register A set to zero. Refer to the CPIM Plus (CP/M Version 3) Operating System System Guide for the definition of the BIOS functions and their register passing and return conventions.

BDOS FUNCTION 59: LOAD OVERLAY

Entry Parameters:
Registers C: 3BH
DE: FCB Address

Returned Value:
Registers A: Error Code
H: Physical Error

Only transient programs with an RSX header can use the Load Overlay function because BDOS Function 59 is supported by the LOADER module. The calling program must have a header to force the LOADER to remain resident after the program is loaded (see Section 1.3).

Function 59 loads either an absolute or relocatable module. Relocatable modules are identified by a filetype of PRL. Function 59 does not call the loaded module.

The referenced FCB must be successfully opened before Function 59 is called. The load address is specified in the first two random record bytes of the FCB, rO and rl. The LOADER returns an error if the load address is less than 100H, or if performing the requested load operation would overlay the LOADER, or any other Resident System Extensions that have been previously loaded.

When loading relocatable files, the LOADER requires enough room at the load address for the complete PRL file including the header and bit map (see Appendix B). Otherwise an error is returned. Function 59 also returns an error on PRL file load requests if the specified load address is not on a page boundary.

Upon return, Function 59 sets register A to zero if the load operation is successful. If the LOADER RSX is not resident in memory because the calling program did not have a RSX header, the BDOS returns with register A set to OFFH and register H set to zero. If the LOADER detects an invalid load address, or if insufficient memory is available to load the overlay, Function 59 returns with register A set to OFEH. All other error returns are consistent with the error codes returned by BDOS Function 20, Read Sequential.

BDOS FUNCTION 60: CALL RESIDENT SYSTEM EXTENSION

Entry Parameters:
Registers C: 3CH
DE: RSX PB Address

Returned Value:
Registers A: Error Code
H: Physical Error

Function 60 is a special BDOS function that you use when you call Resident System Extensions. The RSX subfunction is specified in a structure called the RSX Parameter Block, defined as follows:

```
RSXPB:  db FUNC          ; RSX Function number
        db NUMPARMS      ; Number of word Parameters
        dw PARMETER1     ; Parameter I
        dw PARMETER2     ; Parameter 2
        . . .
        dw PARMETERN     ; Parameter n
```

RSX modules filter all BDOS calls and capture RSX function calls that they can handle. If there is no RSX module present in memory that can handle a specific RSX function call, the call is not trapped, and the BDOS returns OFFH in registers A and L. RSX function numbers from 0 to 127 are available for CP/M 3 compatible software use. RSX function numbers 128 to 255 are reserved for system use.

BDOS FUNCTION 98: FREE BLOCKS

Entry Parameters:
Register C: 62H

Returned Value:
Registers A: Error Flag
H: Physical Error

The Free Blocks function scans all the currently logged-in drives, and for each drive returns to free space all temporarily-allocated data blocks. A temporarily-allocated data block is a block that has been allocated to a file by a BDOS write operation but has not been permanently recorded in the directory by a BDOS close operation. The CCP calls Function 98 when it receives control following a system warm start. Be sure to close your file, particularly any file you have written to, prior to calling Function 98.

In the nonbanked version of CP/M 3, Function 98 frees only temporarily allocated blocks for systems that request double allocation vectors in GENCPM.

Upon return, register A is set to zero if Function 98 is successful. If a physical error is encountered, the Free Blocks function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the Free Blocks function returns to the calling program with register A set to OFFH and register H set to the following physical error code:

04 : Invalid drive error

BDOS FUNCTION 99: TRUNCATE FILE

Entry Parameters:
Registers C: 63H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Extended or Physical Error

The Truncate File function sets the last record of a file to the random record number contained in the referenced FCB. The calling program passes the address of the FCB in register pair DE, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and bytes 33 through 35, rO, rl, and r2, specifying the last record number of the file. The last record number is a 24 bit value, stored with the least significant byte first, rO, the middle byte next, rl, and the high byte last, r2. This value can range from 0 to 262,143, which corresponds to a maximum value of 3 in byte r2.

If the file specified by the referenced FCB is password protected, the correct password must be placed in the first eight bytes of the current DMA buffer, or have been previously established as the default password (see Function 106).

Function 99 requires that the file specified by the FCB not be open, particularly if the file has been written to. In addition, any activated FCBs naming the file are not valid after Function 99 is called. Close your file before calling Function 99, and then reopen it after the call to continue processing on the file.

Function 99 also requires that the random record number field of the referenced FCB specify a value less than the current file size. In addition, if the file is sparse, the random record field must specify a record in a region of the file where data exists.

Upon return, the Truncate function returns a Directory Code in register A with the value 0 if the Truncate function is successful, or OFFH, 255 decimal, if the file is not found or the record number is invalid. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the Truncate function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, the Truncate function returns to the calling program with register A set to OFFH and register H set to one of the following physical or extended error codes:

01	Disk I/O error
02	Read-Only disk
03	Read-Only file
04	Invalid drive error
07	File password error
09	? in filename or filetype field

BDOS FUNCTION 100: SET DIRECTORY LABEL

Entry Parameters:
Registers C: 64H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical or Extended Error

The Set Directory Label function creates a directory label, or updates the existing directory label for the specified drive. The calling program passes in register pair DE the address of an FCB containing the name, type, and extent fields to be assigned to the directory label. The name and type fields of the referenced FCB are not used to locate the directory label in the directory; they are simply copied into the updated or created directory label. The extent field of the FCB, byte 12, contains the user's specificat-on of the directory label data byte. The definition of the directory label data byte is:

bit 7 - Require passwords for password-protected files (Not supported in nonbanked CP/M 3 systems)
6 - Perform access date and time stamping
5 - Perform update date and time stamping
4 - Perform create date and time stamping
0 - Assign a new password to the directory label

If the current directory label is password protected, the correct password must be placed in the first eight bytes of the current DMA, or have been previously established as the default password (see Function 106). If bit 0, the low-order bit, of byte 12 of the FCB is set to 1, it indicates that a new password for the directory label has been placed in the second eight bytes of the current DMA.

Note that Function 100 is implemented as an RSX, DIRLBL.RSX, in nonbanked CP/M 3 systems. If Function 100 is called in nonbanked systems when the DIRLBL.RSX is not resident an error code of 0FFH is returned.

Function 100 also requires that the referenced directory contain SFCBs to activate date and time stamping on the drive. If an attempt is made to activate date and time stamping when no SFCBs exist, Function 100 returns an error code of OFFH in register A and performs no action. The CP/M 3 INITDIR utility initializes a directory for date and time stamping by placing an SFCB record in every fourth entry of the directory.

Function 100 returns a Directory Code in register A with the value 0 if the directory label create or update is successful, or OFFH, 255 decimal, if no space exists in the referenced directory to create a directory label, or if date and time stamping was requested and the referenced directory did not contain SFCBS. Register H is set to zero in both of these cases. If a physical error or extended error is encountered, Function 100 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, Function 100 returns to the calling program with register A set to OFFH and register H set to one of the following physical or extended error codes:

01 Disk I/O error
02 Read-Only disk
04 Invalid drive error
07 File password error

BDOS FUNCTION 101: RETURN DIRECTORY LABEL DATA

Entry Parameters:
Registers C: 65H
E: Drive

Returned Value:
Registers A: Directory Label Data Byte
H: Physical Error

The Return Directory Label Data function returns the data byte of the directory label for the specified drive. The calling program passes the drive number in register E with 0 for drive A, 1 for drive B, and so on through 15 for drive P in a full sixteen drive system. The format of the directory label data byte is shown below:

bit 7 - Require passwords for password protected files
6 - Perform access date and time stamping
5 - Perform update date and time stamping
4 - Perform create date and time stamping
0 - Directory label exists on drive

Function 101 returns the directory label data byte to the calling program in register A. Register A equal to zero indicates that no directory label exists on the specified drive. If a physical error is encountered by Function 101 when the BDOS Error mode is in one of the return modes (see Function 45), this function returns with register A set to OFFH, 25S decimal, and register H set to one of the following:

01 Disk I/O error
04 Invalid drive error

BDOS FUNCTION 102: READ FILE DATE STAMPS AND PASSWORD MODE

Entry Parameters:
Registers C: 66H
DE: FCB Address

Returned Value:
Registers A: Directory Code
H: Physical Error

Function 102 returns the date and time stamp information and password mode for the specified file in byte 12 and bytes 24 through 32 of the specified FCB. The calling program passes in register pair DE, the address of an FCB in which the drive, filename, and filetype fields have been defined.

If Function 102 is successful, it sets the following fields in the referenced FCB:

byte 12 : Password mode field
bit 7 - Read mode
bit 6 - Write mode
bit 4 - Delete mode

Byte 12 equal to zero indicates the file has not been assigned a password. In non-banked systems, byte 12 is always set to zero.

byte 24 - 27 Create or Access time stamp field
byte 28 - 31 Update time stamp field

The date stamp fields are set to binary zeros if a stamp has not been made. The format of the time stamp fields is the same as the format of the date and time structure described in Function 104.

Upon return, Function 102 returns a Directory Code in register A with the value zero if the function is successful, or OFFH, 255 decimal, if the specified file is not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, Function 102 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, Function 102 returns to the calling program with register A set to OFFH and register H set to one of the following physical or extended error codes:

01 Disk I/O error
04 Invalid drive error
09 ? in filename or filetype field

BDOS FUNCTION 103: WRITE FILE XFCB

Entry Parameters:
  Registers C: 67H
    DE: FCB Address

Returned Value:
  Registers A: Directory Code
    H: Physical Error

Ile Write File XFCB function creates a new XFCB or updates the existing XFCB for the specified file. The calling program passes in register pair DE the address of an FCB in which the drive, name, type, and extent fields have been defined. The extent field specifies the password mode and whether a new password is to be assigned to the file. The format of the extent byte is shown below:

FCB byte 12 (ex) : XFCB password mode
bit 7 - Read mode
bit 6 - Write mode
bit 5 - Delete mode
bit 0 - Assign new password to the file

If the specified file is currently password protected, the correct password must reside in the first eight bytes of the current DMA, or have been previously established as the default password (see Function 106). If bit 0 is set to 1, the new password must reside in the second eight bytes of the current DMA.

Upon return, Function 103 returns a Directory Code in register A with the value zero if the XFCB create or update is successful, or OFFH, 255 decimal, if no directory label exists on the specified drive, or the file named in the FCB is not found, or no space exists in the directory to create an XFCB. Function 103 also returns with OFFH in register A if passwords are not enabled by the referenced directory's label. On nonbanked systems, this function always returns with register A = OFFH because passwords are not supported. Register H is set to zero in all of these cases. If a physical or extended error is encountered, Function 103 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, Function 103 returns to the calling program with register A set to OFFH and register H set to one of the following physical or extended error codes:

01 Disk I/O error
02 Read-Only disk
04 Invalid drive error
07 File password error
09 ? in filename or filetype field

BDOS FUNCTION 104: SET DATE AND TIME

Entry Parameters:
Registers C: 68H
DE: DAT Address

Returned Value: none

The Set Date and Time function sets the system internal date and time. The calling program passes the address of a 4-byte structure containing the date and time specification in the register pair DE. The format of the date and time (DAT) data structure is:

byte 0 - 1	Date field
byte 2	Hour field
byte 3	Minute field

The date is represented as a 16-bit integer with day 1 corresponding to January 1, 1978. The time is represented as two bytes: hours and minutes are stored as two BCD digits.

This function also sets the seconds field of the system date and time to zero.

BDOS FUNCTION 105: GET DATE AND TIME

Entry Parameters:
Registers C: 69H
DE: DAT Address

Returned Value:
Register A: seconds
DAT set

The Get Date and Time function obtains the system internal date and time. The calling program passes in register pair DE, the address of a 4-byte data structure which receives the date and time values. The format of the date and time, DAT, data structure is the same as the format described in Function 104. Function 105 also returns the seconds field of the system date and time in register A as a two digit BCD value.

BDOS FUNCTION 106: SET DEFAULT PASSWORD

Entry Parameters:
Registers C: 6AH
DE: Password Address

Returned Value: none

The Set Default Password function allows a program to specify a password value before a file protected by the password is accessed. When the file system accesses a password-protected file, it checks the current DMA, and the default password for the correct value. If either value matches the file's password, full access to the file is allowed. Note that this function performs no action in nonbanked CP/M 3 systems because file passwords are not supported.

To make a Function 106 call, the calling program sets register pair DE to the address of an 8-byte field containing the password.

BDOS FUNCTION 107: RETURN SERIAL NUMBER

Entry Parameters:
Registers C: 6BH
DE: Serial Number Field

Returned Value: Serial number field set

Function 107 returns the CP/M 3 serial number to the 6-byte field addressed by register pair DE.

BDOS FUNCTION 108: GET/SET PROGRAM RETURN CODE

Entry Parameters:
Registers C: 6CH
DE: 0FFFFH (Get) or
Program Return Code (Set)

Returned Value:
Register HL: Program Return Code or (no value)

CP/M 3 allows programs to set a return code before terminating. This provides a mechanism for programs to pass an error code or value to a following job step in batch environments. For example, Program Return Codes are used by the CCP in CP/M 3's conditional command line batch facility. Conditional command lines are command lines that begin with a colon, :. The execution of a conditional command depends on the successful execution of the preceding command. The CCP tests the return code of a terminating program to determine whether it successfully completed or terminated in error. Program return codes can also be used by programs to pass an error code or value to a chained program (see Function 47, Chain To Program).

A program can set or interrogate the Program Return Code by calling Function 108. If re 'ster pair DE = OFFFFH, then the current Program Return Code is returned in register pair HL. Otherwise, Function 108 sets the Program Return Code to the 'value contained in register pair DE. Program Return Codes are defined in Table 3-5.

Table 3-5. Program Return Codes

Code	Meaning
0000 - FEFF	Successful return
FF00 - FFFE	Unsuccessful return
0000	The CCP initializes the Program Return Code to zero unless the program is loaded as the result of program chain.
FF80 - FFFC	Reserved
FFFD	The program is terminated because of a fatal BDOS error.
FFFE	The program is terminated by the BDOS because the user typed a CTRL-C.

BDOS FUNCTION 109: GET/SET CONSOLE MODE

Entry Parameters:
Registers C: 6DH
DE: OFFFFH (Get) or Console Mode (Set)

Returned Value:
Register HL: Console Mode or (no value)

A program can set or interrogate the Console Mode by calling Function 109. If register pair DE = OFFFFH, then the current Console Mode is returned in register HL. Otherwise, Function 109 sets the Console Mode to the value contained in register pair DE.

The Console Mode is a 16-bit system parameter that determines the action of certain BDOS Console I/O functions. The definition of the Console Mode is:

bit 0 = 1 - CTRL-C only status for Function 1 1.
= 0 - Normal status for Function 1 1.

bit 1 = 1- Disable stop scroll, CTRL-S, start scroll, CTRL-Q, support.
= 0-Enable stop scroll, start scroll support.

bit 2 = 1- Raw console output mode. Disables tab expansion for Functions 2, 9 and 111. Also disables printer echo, CTIRL-P, support.
= 0 - Normal console output mode.

bit 3 = 1 - Disable CTRL-C program termination
= 0 - Enable CTRL-C program termination

bits 8,9 -Console status mode for RSXs that perform console input redirection from a file. These bits determine how the RSX responds to console status requests.

bit 8 = 03 bit 9 = 0 - conditional status
bit 8 = 01 bit 9 = 1 - false status
bit 8 = 1, bit 9 = 0 - true status
bit 8 = 11 bit 9 = 1 - bypass redirection

Note that the Console Mode bits are numbered from right to left.

The CCP initializes the Console Mode to zero when it loads a program unless the program has an RSX that overrides the default value. Refer to Section 2.2.1 for detailed information on Console Mode.

BDOS FUNCTION 110: GET/SET OUTPUT DELIMITER

Entry Parameters:
Registers C: 6EH
DE: OFFFFH (Get) or
E: Output Delimiter (Set)

Returned Value:
Register A: Output Delimiter or (no value)

A program can set or interrogate the current Output Delimiter by calling Function 110. If register pair DE = 0FFFFH, then the current Output Delimiter is returned in register A. Otherwise, Function 110 sets the Output Delimiter to the value contained in register E.

Function 110 sets the string delimiter for Function 9, Print String. The default delimiter value is a dollar sign, $. The CCP restores the Output Delimiter to the default value when a transient program is loaded.

BDOS FUNCTION111: PRINT BLOCK

Entry Parameters:
Registers C: 6FH
DE: CCB Address

Returned Value: none

The Print Block function sends the character string located by the Character Control Block, CCB, addressed in register pair DE, to the logical console, CONOUT:. If the Console Mode is in the default state (see Section 2.2.1), Function 111 expands tab characters, CTRL-I, in columns of eight characters. It also checks for stop scroll, CTRL-S, start scroll, CTRL-Q, and echoes to the logical list device, LST:, if printer echo, CTRL-P, has been invoked.

The CCB format is:

byte 0 - 1 Address of character string (word value)
byte 2 - 3 Length of character string (word value)

BDOS FUNCTION 112: LIST BLOCK

Entry Parameters:
Registers C: 70H
DE: CCB Address

Returned Value: none

The List Block function sends the character string located by the Character Control Block, CCB, addressed in register pair DE, to the logical list device, LST.-.

The CCB format is:

byte 0 - 1 Address of character string (word value)
byte 2 - 3 Length of character string (word value)

BDOS FUNCTION 152: PARSE FILENAME

Entry Parameters:
Registers C: 98H
DE: PFCB Address

Returned Value:
Register HL: Return code
Parsed file control block

The Parse Filename function parses an ASCII file specification and prepares a File Control Block, FCB. The calling program passes the address of a data structure called the Parse Filename Control Block, PFCB, in register pair DE. The PFCB contains the address of the input ASCII filename string followed by the address of the target FCB as shown below:

```
PFCB: DW INPUT       ; Address of input ASCII string
      DW FCB         ; Address of target FCB
```

The maximum length of the input ASCII string to be parsed is 128 bytes. The target FCB must be 36 bytes in length.

Function 152 assumes the input string contains file specifications in the following form:

{d:}filename{.typll}{;password}

where items enclosed in curly brackets are optional. Function 152 also accepts isolated drive specifications d: in the input string. When it encounters one, it sets the filename, filetype, and password fields in the FCB to blank.

The Parse Filename function parses the first file specification it finds in the input string. The function first eliminates leading blanks and tabs. The function then assumes that the file specification ends on the first delimiter it encounters that is out of context with the specific field it is parsing. For instance, if it finds a colon, and it is not the second character of the file specification, the colon delimits the entire file specification.

Function 152 recognizes the following characters as detimiters:

space
tab
return
nut]
; (semicolon) - except before password field
= (equal)
< (less than)
> (greater than)
. (period) - except after filename and before filetype
: (colon) - except before filename and after drive
, (comma)
| (vertical bar)
[(left square bracket)
] (right square bracket)

If Function 152 encounters a non-graphic character in the range 1 through 31 not listed above, it treats the character as an error. The Parse Filename function initializes the specified FCB shown in Table 3-6.

Table 3-6. FCB Format

Location	Contents
byte 0	The drive field is set to the specified drive. If the drive is not specified, the default drive code is used. 0 = default, 1 = A, 2 = B.
byte 1-8	The name is set to the specified filename. All letters are converted to upper-case. If the name is not eight characters long, the remaining bytes in the filename field are padded with blanks. If the filename has an asterisk,, *, all remaining bytes in the filename field are filled in with question marks, ?. An error occurs if the filename is more than eight bytes long.
byte 9-11	The type is set to the specified filetype. If no filetype is specified, the type field is initialized to blanks. All letters are converted to upper-case. If the type is not three characters long, the remaining bytes in the filetype field are padded with blanks. If an asterisk, *, occurs, all remaining bytes are filled in with question marks, ?. An error occurs if the type field is more than three bytes long.
byte 12-15	Filled in with zeros.
byte 16-23	The password field is set to the specified password. If no password is specified, it is initialized to blanks. If the password is less than eight characters long, remaining bytes are padded with blanks. All letters are converted to upper-case. If the password field is more than eight bytes long, an error occurs. Note that a blank in the first position of the password field implies no password was specified.
byte 24-31	Reserved for system use.

If an error occurs, Function 152 returns an 0FFFFH in register pair HL.

On a successful parse, the Parse Filename function checks the next item in the input string. It skips over trailing blanks and tabs and looks at the next character. If the character is a null or carriage return, it returns a 0 indicating the end of the input string. If the character is a delimiter, it returns the address of the delimiter. If the character is not a delimiter, it returns the address of the first trailing blank or tab.

If the first non-blank or non-tab character in the input string is a null, 0, or carriage return, the Parse Filename function returns a zero indicating the end of string.

If the Parse Filename function is to be used to parse a subsequent file specification in the input string, the returned address must be advanced over the delimiter before placing it in the PFCB.

End of Section 3

Section 4

Programming Examples

The programs presented in this section illustrate how to use the BDOS functions described in the previous section. The examples show how to copy a file, how to dump a file, how to create or access a random access file, and how to write an RSX program.

4.1 A Sample File-To-File Copy Program

The following program illustrates simple file operations. You can create the program source file, COPY.ASM, using ED or another editor, and then assemble COPY.ASM using MAC. MAC produces the file COPY.HEX. Use the utility HEXCOM to produce a COPY.COM file that can execute under CP/M 3.

The COPY program first sets the stack pointer to a local area, then moves the second name from the default area at 006CH to a 33-byte file control block named DFCB. The DFCB is then prepared for file operations by clearing the current record field. Because the CCP sets up the source FCB at 005CH upon entry to the COPY program, the source and destination FCBs are now ready for processing. To prepare the source FCB, the CCP places the first name into the default FCB, with the proper fields zeroed, including the current record field at 007CH.

COPY continues by opening the source file, deleting any existing destination file, and then creating the destination file. If each of these operations is successful, the COPY program loops at the label COPY until each record is read from the source file and placed into the destination file. Upon completion of the data transfer, the destination file is closed, and the program returns to the CCP command level by lumping to BOOT.

```
                ;       sample file-to-file copy program
                ;       at the ccp level , the command
                ;       copy a:x.y b:u.v
                ;       copies the file named x.y from drive
                ;       a to a file named u.v on drive b.
0000    =       boot    equu    0000h  ; system reb00t
0005    =       bdos    equ     0005h  ; bdos entry P0int
005c    =       fcbl    equ     00Sch  ; first file name
005c    =       sfcb    equ     fcbl   ; source fcb
006c    =       fcbZ    equ     00Bch  ; second file name
0080    =       dbuff   equ     0080h  ; default buffer
0100    =       tpa     equ     0100h  ; beginning 0f tpa
0003    =       printf  equ     9      ; Print buffer func*
000f    =       openf   equ     15     ; Open file func*
0010    =       closef  equ     16     ; close file func*
0013    =       deletef equ     19     ; delete file func*
0014    =       readf   equ     20     ; sequential  read
0015    =       writef  equ     21     ; sequential write
0016    =       makef   equ     22     ; make file func*
                ;
0100                    org     tpa    ; beginning of tpa
0100 311b02             lxi     sp,stack; local stack
                ;
                ;       move second file name to dfcb
0103    0el0            mvi     c,16   ; half an fcb
0105    116c00          lxi     d,fcb2 ;        source of move
0108    2lda0l          lxi     h,dfcb ;        destination fcb
0l0b    la      mfcb:   ldax    d      ;        source fcb
0l0c    13              inx     d      ;        ready next
0l0d    77              m0v     m,a    ; dest fcb
0l0e    23              inx     h      ;        ready next
0l0f    0d              dcr     c      ; count l6...0
0110    c20b0l          jnz     mfcb   ;        loop 16 times
                ;       name has been moved, zero cr
0113    af              xra     a      ;        a = 00h
0114    32fa01          sta     dfcbcr ;        current  rec = 0
```

```
                    ;       source and destination fcbs  ready
                    ;
0117    115c00              lxi     d,sfcb  ; source file
0lla    cd690l              call    0pen    ; error if 255
0lid    118701              lxi     d,nofile; ready message
0120    3c                  inr     a       ; 255 becomes 0
0121    cc6l0l              cz      finis   ; done if no file
                    ;
                    ;       source file open, prep destination
0l24     llda0l             lxi     d ,dfcb ; destination
0127    cd7301              call    delete  ; remove if present
0lZa    llda0l              lxi     d,dfcb  ; destination
0lZd    cdB20l              call    make    ; create the file
0130    119601              lxi     d,nodir ; ready message
0133    3c                  inr     a       ; 255 becomes 0
0134    cc6l0l              cz      finis   ; done if no dir space
                    ;
                    ;       source file open, dest file open
                    ;       copy until end of file on source
0137    115c00copy:         lxi     d,sfcb  ; source
013a    cd780l              call    read    ; read next  record
013d    b7                  ora     a       ; end of file?
013e    c25101              Jnz     eofile  ; skip write if s0
                    ;       not end of file, write the record
0141    llda0l              lxi     d,dfcb  ; destination
0144    cd7d0l              call    write   ; write  record
0147    11a901              lxi     d,space; ready message
014a    b7                  ora     a       ; 00 if write ok
014b    c46101              cnz     finis   ; end if so
014e    c33701              Jmp     copy    ; loop until eof
                    ;
                    eofile: ; end of file, close destination
0151    llda0l              lxi     d,dfcb  ; destination
0154    cd6e0l              call    close   ; 255 if error
0157    2lbb0l              lxi     h,wrprot; ready message
015a    3c                  inr     a       ; 255 becomes 00
015b    cc6l0l              cz      finis   ; should not happen
                    ;
                    ;       copy operation complete, end
0lSe llcc0l                 lxi     d,normal ;  ready message
                    ;
                    finis:  ; write  message          given by de,  reboot
0161    0e09                mvi     c,printf
0163    cd0500              call    bdos    ; write message
0166    c30000              Jmp     boot    ;  reboot system4-3
```

```
            ;         system interface subroutines
            ;         (all  return directly from bdos)
0169 0e0f         open:   mvi     c ,openf
016b c30500               Jmp     bdos
016e 0el0         close:  mvi     c,closef
0170 c30500               Jmp     bdos
0173 0e13         delete: mvi     c ,deletef
0175 c30500               Jmp     bdos
0178 0e14         read:   mvi     C ,readf
017a c30500               Jmp     bdos
017d 0e15         write:  mvi     c ,writef
017f c30500               Jmp     bdos
0182 0e16         maKe:   mvi     c,makef
0184 c30500               Jmp     bdos
                  ; console       messages
0187 6e6f20f nofile:  db      'no source file$'
0196 6e6f209 nodir:   db      'no directory space$'
01a9 5f7s74f  space:  db      'out of data space$'
0lbb 7772695 wrprot: db       'write protected?$'
0lcc 636f700   normal:        db      'copy complete$'
                  ;       data    areas
0lda              dfcb:   ds      33      ;       destination fcb
0lfa =            dfcbcr  equ     dfcb+32         ;       current  record
0lfb                      ds      32      ; 16 level stack
                  stack:
021b              end
```

Note that this program makes several simplifications and could be enhanced. First, it does not check for invalid filenames that could, for example, contain ambiguous references. This situation could be detected by scanning the 32-byte default area starting at location 005CH for ASCII question marks. To check that the filenames have, in fact, been included, COPY could check locations 005DH and 006DH for nonblank ASCII characters. Finally, a check should be made to ensure that the source and destination filenames are different. Speed could be improved by buffering more data on each read operation. For example, you could determine the size of memory by fetching FBASE from location 0006H, and use the entire remaining portion of memory for a data buffer. You could also use CP/M 3's Multi-Sector I/O facility to read and write data in up to 16K units.

4.2 A Sample File Dump Utility

The following dump program reads an input file specified in the CCP command line, and then displays the content of each record in hexadecimal format at the console. DUMP Program reads input file and displays hex data

```
              ; DUMP program reads input file and displays hex data
              ;
0100                        org     l00h
0005    =       bd0s        equ     0005h  ;bdos entry point
0001    =       c0ns        equ     1      ;read console
0002    =       typef       equ     2      ;type function
0009    =       printf      equ     9      ;buffer print entry
000b    =       brkf        equ     11     ;break key functi0n  (true if char
000f    =       openf       equ     15     ;file open
0014    =       readf       equ     20     ;read function
005c    =       fcb         equ     Sch    ;file control block address
0080    =       buff        equ     80h    ;input disk buffer address
              ;         non graphic characters
000d    =       cr          equ     0dh    ;carriage  return
000a    =       lf          equ     0ah    ;line feed
                        ;   file    control block definitions
005c    =       fcbdn       equ     fcb+0  ;disk name
005d    =       fcbfn       equ     fcb+1  ;file name
0065    =       fcbft       equ     fcb+B  ;disk file tyPe   (3 characters)
0068    =       fcbrl       equ     fcb+12 ;file's current    reel number
00Gb    =       fcbrc       equ     fcb+15 ;file's  record count  (0 to 128)
007c    =       fcbcr       equ     fcb+32 ;current  (next)         record number (0
007d    =       fcbln       equ     fcb+33 ;fcb length
                        ;   set up stack
0100    210000              lxi     h,0
0103    39                  dad     sp
                        ;   entry stack pointer in hi from the ccp
0104    221502              shld    oldsp
                        ;   set sp to local stack area (restored at finis)
0107    315702              lxi     sp,stktop
                        ;   read and Print successive buffers
0l0a    cdcl0l              call    setup   ;set up input file
0l0d    feff                cpi     255     ;255 if file not present
0l0f    c2lb0l              jnz     openok   ;skip if open is ok
                        ;
```

```
                    ;       file not there, give error message and  return
0112   11f301               lxi     d,opnmsg
0115   cd9c0l               call    err
0118   c35101               jmp     finis   ;to  return
                    openok:         ;open operation ok, set buffer index to end
0lib   3e80                 mvi      a,80h
0lid   321302               sta     ibp      ;set buffer pointer to 80h
                    ;       hl contains next address to print
0120   210000               lxi     h,0      ;start with 0000
                    gloop:
0123   e5                   push     h       ;save line position
0124   cda20l               call     gnb
0127   el                   pop      h       ;recall line position
0128   da5l0l               jc       finis   ;carry set by gnb if end file
012b   47                   mov      b,a
                    ;       print   hex values
                    ;       check   for line fold
012c   7d                   mov      a,l
012d   e80f                 ani      0fh     ;check low 4 bits
012f   c24401               jnz      nonum
                    ;       print   line number
0132   cd7201               call     crlf
                    ;       check for break key
0135   cd590l               call    break
                    ;       accum lsb = 1 if character ready
0138 0f                     rrc              ;into carry
0139   da5l0l               jc      finis    ;do not print any more
 013c 7c                    mov     a,h
013d   cdBf0l               call    phex
0140   7d                   mov     a,l
0141   cd8f0l               call    phex
                    nonum:
0144   23                   inx     h        ;to next line number
0145   3e20                 mvi     a,' '
0147   cd650l               call    pchar
014a   78                   mov     a,b
014b   cd8f0l               call    phex
014e   c32301               jmp     gloop
```

```
                ;
                finis:
                ;       end of dump
0151  cd7201            call    crlf
0154  2a1502            lhld    oldsp
0157  f9                sphl
                ;       stack pointer contains ccp's stack location
0158 c9                 ret             ;to the ccp
                ;       subroutines
                break:  ;check break key  (actually any key will do)
0159  e5d5c5            push h!push d!  push b; environment saved
015c  0e0b              mvi     c,brkf
0l5e  cd0500            call    bdos
0161  cidlel            pop b!  pop d!  pop h; environment  restored
0164  c9                ret
                pchar:  ;print a character
0165  e5d5c5            push h!  push d!  push b; saved
0188  0e02              mvi     c,typef
016a  5f                mov      e ,a
016b  cd0500            call    bdos
016e  cidlel            pop b!  pop d!  pop h;  restored
0171  c9                ret
                crlf:
0172  3e0d              mvi     a,cr
0174  cd6501            call    pchar
0177  3e0a              mvi     a,lf
0179  cdBS0l            call    pchar
017c  c9                ret
                pnib:   ;print  nibble  in  reg a
017d  e60f              ani      0fh      ;low 4 bits
017f  fe0a              cpi      10
0181  d28901                    jnc      p10
                ;       less    than or equal to 9
0184  c630              adi      '0'
0188  c38b01            jmp      prn
                ;       greater 0r equal t0 10
0189  c637      pl0:    adi     'a' - 10
018b  cd6501    prn:    call    pchar
018e  c9                ret
```

```
                phex:   ;Print hex char in  reg a
018f    f5              push    psw
0190    0f              rrc
0191    0f              rrc
0192    0f              rrc
0193    0f              rrc
0194    cd7d0l          call    pnib    ;print nibble
0197    fi              pop     pSW
0198    cd7d0l          call    pnib
018b    c9              ret
                err:    ;Print err0r message
                ;       d,e addresses message ending With"$"
019c    0e09            mvi     c,printf        ;print buffer functi0n
019e    cd0S00                  call    bdos
0lal    c9              ret
                gnb     ;get next byte
01a2 3a1302             lda     ibp
0la5 fe80               cPi     80h
01a7 c2b301             jnz     g0
                ;       read another buffer
0laa    cdce0l          call    diskr
0lad    b7              ora     a       ;zero value if  read ok
0lae    cab30l          jz      g0      ;for another byte
                ;       end of data,            return With carry Set for eof
0lbi    37              stc
01b2    c9              ret
                g0:     ;read the byte at buff+reg a
01b3    5f              mov     e,a     ;ls byte of buffer index
01b4    1600            mvi     d,0     ;double precision index to de
01b6    3c              inr     a       ;index=index+l
01b7    321302                  sta     ibp     ;back to memory
                ;       Pointer is incremented
                ;       save the current file address
0lba    218000                  lxi     h,buff
0lbd    19              dad     d
                ;       abs0lute character address is in hl
0lbe    7e              mov     a,m
                ;       byte is  in the accumulator
0lbf    b7              ora     a       ;reset carry bit
0lc0    c9              ret
```

```
                    setup:  ;set up file
                    ;       open the file for input
01c1 af                     xra     a       ;zero to accum
01c2 327c00                 sta     fcbcr   ;clear current rec0rd
0lc5 115c00                 lxi     d,fcb
01c8 0e0f                   mvi     c,openf
0lca cd0500                 call    bdos
                    ;       255 in accum if open error
0lcd c9 ret
                    diskr:  ;read disk file  record
0lce    e5d5c5              Push h!Push d!  Push b
0ldI    115c00              lxi     d,fcb
01d4    0e14                mvi     c,readf
01d6    cd0500              call    bdos
01d9    c1d1e1              Pop b!  Pop d!  Pop h
0ldc    c9                  ret
                    ;       fixed message area
0ldd 46494c0 signon: db     'file dumP version 2.0$'
01f3 0d0a4e0 opnmsg: db     cr,lf ,'no input file Present on disk$'
                    ;       variable area
0213                        ibp:    ds      2       ;input buffer Pointer
0215                        oldsp:  ds      2       ;entry sP value from ccP
                    ;       stack area
0217                        ds      64      ;reserve 32 level stack
                    stktop:
0257                        end
```

4.3 A Sample Random Access Program

This example is an extensive but complete example of random access operation. The following program reads or writes random records upon command from the terminal. When the program has been created, assembled, and placed into a file labeled RANDOM.COM, the CCP level command

A>RANDOM X.DAT

can start the test program. In this case, the RANDOM program looks for a file X.DAT and, if it finds it, prompts the console for input. If X.DAT is not found, RANDOM creates the file before displaying the prompt. Each prompt takes the form:

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form:

nW nR nF Q

where n is an integer value in the range 0 to 262143, and W, R, F, and Q are simple command characters corresponding to random write, W, random read, R, random write with zero fill, F, and quit processing, Q. If you enter a W or F command, the RANDOM program issues the prompt:

type data:

You then respond by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If you enter an F command, the RANDOM program fills previously unallocated data blocks with zeros before writing record n. If you enter the R command, RANDOM reads record number n and displays the string value at the console. If you enter the Q command, the X.DAT file is closed, and the program returns to the console command processor. In the interest of brevity, the only error message is:

err , try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label ready where the individual commands are interpreted. The program uses the default file control block at 005CH and the default buffer at 0080H in all disk operations. The utility subroutines that follow contain the principal input line processor, called readc. This particular program shows the elements of random access processing and can be used as the basis for further program development.

```
;*****************************************
;*                                       *
;*   sample  random access Program for cP/m 3       *
;*                                       *
;*****************************************
0100            org     l00h            ;base of tpa
0000 =reboot    equ     0000h           ;system reboot
0005 =bdos      equ     0005h           ;bdos entry Point
0001   -        coninp equ     1        ;console input function
0002   -        conout equ     2        ;console outPut functo0n
0009   -        pstring equ    9        ;Print string until  '$'
000a   =        rstring equ    10       ;read console buffer
000c   -        version equ    12       ;return version number
00of   =        openf  equ     15       ;file open function
0010   -        closef equ     16       ;close function
0016   -        makef  equ     22       ;make file function
0021   -        readr  equ     33       ;read  random
0022   -        Writer equ     34       ;write  random
0028   -        Wrtrzf equ     40       ;Write  random zero fill
0098   -        parsef equ     152      ;parse function
005c   -        fcb    equ     00Sch    ;default file control block
007D   -        ranrec equ     fcb+33   ;random record Position
007F   =        ran0vf equ     fcb+35   ;high order (overflow)  byte
0080   =        buff   equ     0080h    ;buffer address
000D =          or     equ     0dh      ;carriage  return
000a   =        1f     equ     0ah      ;line feed
```

```
                ;*********************************
                ;*                               *
                ;* load SP, Set-up file for random access *
                ;*                               *
                ;*********************************
0100 313703             lxi     sp,stack
                ;       version 3.1?
0103    0E0c            mvi     c,version
0105    cD0500          call    bdos
0108    FE20            cp      31h     ;version 3.1 or better?
0i0a    D21601          jnc     versok
                ;       bad version, message and go back
010D    118102          lxi     d,badver
0110    cD3102          call    Print
0113    c30000          jmp     reboot
                versok :
                ;       correct version for random access
0116    QEof            mvi     c,openf;open default fcb
0118    3a5D00rdname:   lda     fcb+1
011b    FE20            cPi     ' '
011d    c22c01          jnz     opfile
0120    11E002          lxi     d,entmsg
0123    cD3102          call    Print
0126    cD2002          call    Parse
0129    c31801          jmP     rdname
012c    iisc00  opfile: lxi     d,fcb
012F    cd0so0          call    bdos
0132    3c              inr     a       ;err 255 becomes zero
0133    c24501          jnz     ready
                ;       cannot open file, so create it
0136    0E16            mvi     c,makef
0138    115c00          lxi     d,fcb
013b    cd0so0          call    bdos
013E    3c              inr     a       ;err 255 becomes zero
013F    c24b01          jnz     ready
                ;       cannot create file, directory full
0142    iia00z          lxi     d,nospace
0145    cd3102          call    print
0148    c30000          jmp     reboot  ;back to ccP
```

```
                ;****************************************
                ;*                                      *
                ;*      loop back to "ready" after each command     *
                ;*                                      *
                ;****************************************
                ready:
                ;       file is  ready for processing
014b    cD3C02          call    readcom         ;read next command
014E    227D00          shld    ranrec  ;store input  record*
0151    217F00          lxi     h,ran0vf
0154    71              mov     m,c     ;set ranrec high byte
0155    FE51            cpi     'Q'     ;quit?
0157    C26901          jnz     notq
                ;       quit processing,                close file
0isa    0E10            mvi     c,closef
01SC    115C00          lxi     d,fcb
015F    CD0500          call    bdos
0162    3C              inr     a       ;err 255 becomes 0
0163    CaFF0ljz        error   ;error meSsage ' retry
0166    c30000          jmp     reboot  iback to ccP
                ;
                ;****************************
                ;* end of quit command, pr0cess write          *
                ;*      *
                ;************************************************
                notq:
                ;       not the quit command,  random Write?
0169 FES7               cpi     'W'
016b C29C01             jnz     notw
                ;       this is a random Write, fill buffer until cr
016E    11b302lxi        d,datmsg
0171    CD3102          call     Print  ;data Prompt
0174    0E7F            mvi      c,127 ;UP to 127 characters
0176    218000          lxi      h,buff;destination
                rloop:  ;read   next character to buff
0179    C5              push     b      ;save counter
017a    E5              push     h      ;next destination
017b    CD0802          call     getchr;character to a
017E    El              pop      h      +;irestore counter
017F    C1              pop      b      ;restore next to fill
0180    FE0D            cpi      Cr     ;end of line?
0182    Ca8b0l          jz       erloop
```

```
                ;         not end; Store character
0185    77                mov     m,a
0186    23                inx      h       +;inext to fill
0187    0D                dcr      c       +;icounter goes down
0188    C27901                    jnz      rloop  +;iend of buffer?
                erloop:
                ;         end of  read loop, Store 00
018b    3600              mvi      m,0
                ;         write the  record              to selected  record number
018D  0E22                mvi     c,Writer
018F   115C00             lxi     d,fcb
0192   CD0so0             call    bdos
0195   b7                 ora     a        +;ierror code zero?
0196   C2FF0I             jnz     error   +;imessage if not
0199   C34b01             jmp     ready   +;ifor another record


                ; **********************************************
                ;*                                            *
                ;* end of Write command, pr0cess Write  random zero fill   *
                ;*                                            *
                ; **********************************************
                notw:
                ;         not the quit command;  random Write zero fill?
019C FE46                 cpi     'F'
019E C2CF0I               jnz     notf

                ;         this  is a random Write; fill buffer until cr
01a1    11b302lxi         d,datmsg
01a4    CD3102            call    print   ;data Prompt
01a7    0E7F              mvi     c,127   +;iUP to 127 characters
01a9    218000            lxi     h,buff  ;destination
                rloop1:  ;iread next character to buff
0laC C5                   push    b        +;isave counter
0laD E5                   push    h        +;inext destination
0laE CD0802               call    getchr  +;icharacter to a
0lbI El                   pop     h        +;irestore counter
01b2 C1                   pop     b        +;irestore next to fill
01b3 FE0D                 cpi     cr       +;iend of line?
01b5 CabE0l               jz      erloop1
                ;         not end , store character
01b8 77                   mov     m,a
01b9 23                   inx     h        ;next to fill
0lba 0D                   dcr     c        ;counter goes down
0lbb C2aC01               jnz     rloopl  ;end of buffer?
```

```
                erloop1 :
                ;       end of  read loop, Store 00
0lbE 3600               mvi     m,0

                ;       Write the  record to selected  record number
0lc0   0E28             mvi     c,wrtrzf
01C2   115C00           lxi     d,fcb
01C5   CD0500           call    bdos
01C8   b7               ora     a       ;error code zero?
01C9   C2FF0ljnz        error   ;message if not
01CC   C34b01           jmp     ready  +;ifor another record

                ;***********************************
                ;*                                 *
                ;* end of Write commands; pr0cess read   *
                ;*                                 *
                ;***********************************
                notf :
                ;       not a Write command, read record?
0ICF FES2               cpi     'R'
0lDi C2FF01             jnz     error   ;skip if not
                ;       read    random record
01D4   0E21             mvi     c;readr
01DB   115C00           lxi     d,fcb
01D9   CD0500           call    bdos
0lDC   b7               ora     a      +;ireturn code 00?
0IDD   C2FF0I           jnz     error
                ;       read    was Successful , Write to c0nsole
0lE0   CD1502           call    crlf    ;new line
01E3 0E80               mvi      c,128   ;max 128 characters
0IE5 218000             lxi     h,buff  +;inext to get
                wloop :
01E8 7E                 mov      a,m     ;next character
01E9 23                 inx     h       ;next to get
0lEa E67F               ani     7fh     ;mask Parity
0lEC Ca4b0l             jz      ready   ;for another command if 00
0lEF C5                 push     b      +;isave counter
0lF0 E5                 push     h      +;isave next to get
0lFl FEZ0               cpi     ' '     ;graphic?
01F3 D40E02             cnc     putchr  +;iskip 0utput if not
0lFB El                 pop      h
01F7 C1                 pop      b
01F8 0D                 dcr     c       +;icount=count-l
01F9 C2E801             jnz     wloop
0lFC C34b01             jmp      ready
```

```
          ;***************************************
          ;*                                     *
          ;* end of  read command , all errors end-up here      *
          ;*                                     *
          ;***************************************
                    e r ror :
01FF  11bF02        lxi     d,errmsg
0202  CD3102        call    Print
0205  C34b01        jmp     ready

          ;****************************
          ;*                          *
          ;* utility subroutines for console I/O  *
          ;i*                         *
          ;****************************
                    getchr:
                              ;read next console character to a
0208  0E0l          mvi     c,coninp
020a  CD0so0        call    bdos
020D  C9            ret
                    putchr:
                              ;write character  a to console
020E  0E02          mvi     c,conout
0210  5F            mov     e ,a      ;character to Send
0211  CD0500        call    bdos      +;isend character
0214  C9            ret
                    ;
                    crlf:
                              ;sen carriage return line feed
0215  3E0D          mvi     a,cr      ;carriage  return
0217  CD0E02        call    putchr
021a  3E0a          mvi     a,lf      ;line feed
021C  CD0E02        call    putchr
021F  C9            ret
                    ;
                    parse :
                              ;read and parse filespec
0220  11F102        lxi     d,c0nbuf
0223  0E0a          mvi     c,rstring
0225  CD0500        call    bdos
0228  111303        lxi     d,pfncb
022b  0E98          mvi     c,parsef
022D  CD0500        call    bdos
0230  C9            ret
```

```
                print:
                        ;print the buffer addressed by de Until $
0231   D5               push    d
0232   CD1502           call    crlf
0235   Dl               pop     d       ;new line
0236   0E09             mvi     c,pstring
0238   CD0so0           call    bdos    ;print the String
023b   C9               ret
                readcom:
                        ;iread the next command line to the conbuf
023C   11D102           lxi     d,Prompt
023F   CD3102           call    print   ;command?
0242   0E0a             mvi     c,rstring
0244   11F102           lxi     d,conbuf
0247   CD0500           call    bdos    ;read command line
                ;       command         line  iS present, scan it
024a   0E00             mvi     c,0     ;Start With 00
024C   210000           lxi     h,0     ;            0000
024F   11F302           lxi     d,c0nlin;icommand line
0252   la       readc:  idax    d       ;inext command character
0253   13               inx     d       ;ito next command Position
0254   b7               ora     a       ;cannot be end of command
0255   C8               rz
                ;       not zero, numeric?
0256   D630             sui     '0'
0258   FE0a             cpi     10      +;icarry if numeric
025a   D27902           jnc     endrd
```

```
                 ;       add-in  next    digit
025D  F5                 push    psw
025E  79                 mov     a,c          ;value in ahl
025F  29                 dad     h
0260  8F                 adc     a            +;i*2
0261  F5                 push    a            +;isave value * 2
0262  E5                 push    h
0263  29                 dad     h            ;*4
0264  8F                 adc     a
0265  29                 dad     h            ;*8
0266  8F                 adc     a
0267  C1                 pop     b            +;i*2 + *8 = *10
0268  09                 dad     b
0269  C1                 pop     b
026a  88                 adc     b
026b  C1                 pop     b            +;i+digit
026C  48                 mov     c,b
026D  0600               mvi     b,0
026F  09                 dad     b
0270  CE00               aci     0
0272  4F                 mov     c,a
0273  D25202             jnc     readc
0276  C33C02             jmp     readcom
                 endrd :
                 ;       end of  read    , restore value in a
0279  C630               adi     '0'          ;command
027b  FEGI               cpi     ' a '        ;itranslate case?
027D  DB                 rc
                 ;       lower case , mask lower case bits
027E  E65F               ani     101$1lllb
0280  C9                 ret                  ;return with value in chl

                 ;***********************************
                 ;*                                 *
                 ;* string data area for console messages *
                 ;*                                 *
                 ;***********************************
                 badver:
0281  736F727279 db      ' Sorry, you need cP/m version 3$'
                 nospace :
02a0  6E6F206469 db      'n0 directory space$'
                 datmsg :
02b3  7479706520 db      'type data: $'
                 errmsg :
02bF  6572726F72 db      ' error, try again,$'
```

```
                Prompt:
02D1  6E65787420 db      'next command? $'
                entmsg :
02E0  656E746572 db      'enter filename: $'
                ;*****************************
                ;*                           *
                ;* fixed and variable data area *
                ;*                           *
                ;*****************************
02F1  21        conbuf:     db    conlen ;length of console buffer
02F2            consiz: ds  1            ;resulting size after read
02F3            conlin: ds  32           ;length 32 buffer
0021  =         conlen equ  $-consiz
                pfncb :
0313 F302            dW     conlin
0315 5C00            dw     fcb
0317                 ds     32           ;i16 level Stack
                stack :
0337                 end
```

You could make the following major improvements to this program to enhance its operation. With some work, this program could evolve into a simple data base management system. You could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. You could develop a program called GETKEY that first reads a sequential file and extracts a specific field defined by the operator. For example, the command

GETKEY NAMES*DAT LASTNAME 10 20

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting at position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list and writes a new file, called LASTNAME.KEY. This list, sometimes called an inverted index, is an alphabetical list of LASTNAME fields with their corresponding record numbers.

You could rename the program shown above to QUERY, and modify it so that it reads a sorted key file into memory. The command line might appear as

QUERY NAMES.DAT LASTNAME.KEY

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Because the LAST-NAME.KEY list is sorted, you can find a particular entry quickly by performing a binary search, similar to looking up a name in the telephone directory. Start at both ends of the list and examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You will quickly reach the item you are looking for, in log2(n) steps, where you will find the corresponding record number. Fetch and display this record at the console as the program illustrates.

At this point, you are just getting started. With a little more work, you can allow a fixed grouping size, which differs from the 128-byte record shown above. You can accomplish this by keeping track of the record number as well as the byte offset within the record. Knowing the group size, you can randomly access the record containing the proper group, offset to the beginning of the group within the record, and read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing Boolean expressions that compute the set of records that satisfy several relationships, such as a LAST-NAME between HARDY and LAUREL and an AGE less than 45. Display all the records that fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well.

4.4 Construction of an RSX Program

This section describes the standard prefix of a Resident System Extension (RSX) and illustrates the construction of an RSX with an example. (See Section 1.6.4 for a discussion of how RSXs operate under CP/M 3.) RSX programs are usually written in assembler, but you can use other languages if the interface between the language and the calling conventions of the BDOS are set up properly.

4.4.1 The RSX Prefix

The first 27 bytes of an RSX program contain a standard data structure called the RSX prefix. The RSX prefix has the following format:

```
serial:
  d b     0,0,0,0,0,0
start:.
        jmp         ftest                 ; start of Program
n e x t :
        db          0c3h                  ; jump instruction to
        dw           0                    ; next module in line
Prev:
        dw           0                    ; Previous module
remove:
        db          0ffh                  ; remove flag
nonbank:
        db          0                     ; nonbank flag
name:
        db          '123456713'           ; any B-character name
loade r :
        db          0                     ; loader flag
        db          0,0                   ;  reserved area
```

The only fields of the RSX prefix that you must initialize are the remove: flag, the nonbank: flag, and the name: of the RSX.

For compatibility with previous releases of CP/M, the serial: field of the prefix is set to the serial number of the operating system by the LOADER module when the RSX is loaded into memory. Thus, the address in location 6 locates the byte following the serial number of the operating system with or withou; RSXs in memory.

The start: field contains a jump instruction to the beginning of the RSX code where the RSX tests to see if this BDOS function call is to be intercepted or passed on to the next module in line.

The next: field contains a jump instruction to the next module in the chain or the LOADER module if the RSX is the oldest one in memory. The RSX program must make its own BDOS function calls by calling the next: entry point.

The prev: field contains the address of the preceding RSX in memory or location 5 if the RSX is the first RSX in the chain.

The remove: field controls whether the RSX is removed from memory by the next call to the LOADER module via BDOS function 59. If the remove: flag is OFFH, the LOADER removes the RSX from memory. Note that the CCP always calls the LOADER module during a warm start operation. An RSX that remains in memory past warm start because its remove: flag is zero, must set the flag at its termination to ensure its removal from memory at the following warm start.

The nonbank: field controls when the RSX is loaded. If the field is OFFH, the LOADER only loads the module into memory on nonbanked CP/M 3 systems. Otherwise, the RSX is loaded into memory under both banked and nonbanked versions of CP/M 3.

The loader: flag identifies the LOADER RSX. When the LOADER module loads an RSX into memory, it sets this prefix flag of the loaded RSX to zero. However, the loader: flag in the LOADER's prefix contains OFFH. Thus, this flag identifies the last RSX in the chain, which is always the LOADER.

4.4.2 Example of RSX Use

These two sample programs illustrate the use of an RSX program. The first program, CALLVERS, prints a message to the console and then makes a BDOS Function 12 call to obtain the CP/M 3 version number. CALLVERS repeats this sequence five times before terminating. The second program, ECHOVERS, is an RSX that intercepts the BDOS Function 12 call made by CALLVERS, prints a second message, and returns the version 0031H to CALLVERS. Although this example is simple, it illustrates BDOS function interception, stack swapping, and BDOS function calls within an RSX.

```
                ; CALLVERS program
0005   =        bdos     equ    5          ;       entrY point for BDOS
000B   =        prtstr   equ    B          ;       print string funCtion
000C   =        vers     equ    12         ;       get version function
000D   =        cr       equ    0dh        ;       carriage  return
000A   =        lf       equ    0ah        ;       line feed
0100                     org    l00h
0100   1605              mvi    d,5        ;       Perform 5 times
0102   DS       loop:    push   d          ;       save counter
0103   0E0B              mvi    c,Prtstr
0105   111e01            lxi    d,call$msg        ;       Print call message
oioe   CD0500            call   bdos
0l0B   0E0C              mvi    c,vers
0l0D   CD0500            call   bdos       ;       try to get version *
                                           ;       CALLVERS will  intercept
0110   7D                mov    a,l
0111   323401            sta    curvers
0114   d1                pop    d
0115   15                dcr    d          ;       decrement counter
0116   C20201            Jnz    loop
0119   0E00              mvi    c,0
0llB   C30500            JmP    bdos
                         call $msg:
011C   0D0A2A2A2A                          db      cr,lf,'****       CALLVERS **** $'
0134   00       curvers db      0
0135                     end
; ECHOVERS RSX
000B   =        pstring equ     9          ; string Print function
000D   =        cr       equ    0dh
000A =          lf       equ    0ah
                ;               RSX PREFIX STRUCTURE
0000   0000000000               db     0,0,0,0,0,0     ;       room for serial number
0006   C3lB00                   Jmp    ftest    ;       begin of Program
0005   C3       next:    db     0c3H       ;       Jump
000A   0000              dw     0          ; next module in line
000C   0000     prev:    dw     0          ;       previous module
000c   FF       remov: db       0ffh       ;       remove flag set
000F   00       nonbnk:         db     0
0010   4543484F56               db     'ECHOVERS'
0018   000000                   db     0,0,0
```

```
          ftest:                    ;     is this function 12?
001B  79              mov   a,c
001C  FE0C            cpi   12
001E  CA2400          Jz    begin   ;     Yes - intercept
0021  C30500          Jmp   next    ;     some other function
          begin:
0024  210000          lxi   h,0
0027  39              dad   sp      ; save stack
0028  225400          shld  ret$stack
002B  317600          lxi   sp,loc$stack
002E  0E05            mvi   c,pstring
0030  113E00          lxi   d,test$msg      ; Print message
0033  CD0900          call  next    ; call BDDS
0036  2A5400          lhld  ret$stack       ;     restore user stack
0039  F9              sphl
003A  213100          lxi   h,0031h         ; return version number
003D  C9              ret
          test$msg:
003E  0D0A2A2A2A db   cr,lf,'**** ECHDVERS ****$'
          ret$stack:
0054  0000            dw    0
0056                  ds    32              ; 16 level stack
          loc$stack :
0076                  end
```

You can prepare the above programs for execution as follows:

1. Assemble the CALLVERS program using MAC as follows:
   MAC CALLVERS
2. Generate a COM file for CALLVERS with HEXCOM:
   HEXCOM CALLVERS
3. Assemble the RSX program ECHOVERS using RMAC:
   RMAC ECHOYERS
4. Generate a PRL file using the LINK command:
   LINK ECHOVERS [OPI
S. Rename the PRL file to an RSX file:
   RENAME ECHOVERS*RSX=ECHOVERS.PRL
6. Generate a COM file with an attached RSX using the GENCOM command:
   GENCOM CALLVERS ECHOVERS
7. Run the CALLVERS.COM module:
   CALLVERS
   The message
   **** CALLVERS
   followed by the message
   **** ECHOVERS ****
   appears on the screen five times if the RSX program works.

End of Section 4

Appenix A

System Control Block

The System Control Block (SCB) is a CP/M 3 data structure located in the BDOS. CP/M 3 uses this region primarily for communication between the BDOS and the BIOS. However, it is also available for communication between application programs, RSXS, and the BDOS. Note that programs that access the System Control Block are not version independent. They can run only on CP/M 3.

The following list describes the fields of the SCB that are available for access by application programs and RSXS. The location of each field is described as the offset from the start address of the SCB (see BDOS Function 49). The RW/RO column indicates if the SCB field is Read-Write or Read-Only.

Table A-1. SCB Fields and Definitions

Offset	RW/RO	Definition
00 - 04	RO	Reserved for system use.
05	RO	BDOS Version Number.
06 - 09	RW	Reserved for user use. Use these four bytes for your own flags or data.
0A - 0F	RO	Reserved for system use.
10 - 11	RW	Program Error Return Code. This 2-byte field can be used by a program to pass an error code or value to a chained program. CP/M 3's conditional command facility also uses this field to determine if a program executes successfully. The BDOS Function 108 (Get/Set Program Return Code) is used to get/set this value.
12 - 19	RO	Reserved for system use.

Table A-1. (continued)

Offset	RW/RO	Definition
1A	RW	Console Width. This byte contains the number of columns, characters per line, on your console relative to zero. Most systems default this value to 79. You can set this default value by using the GENCPM or the DEVICE utility. The console width value is used by the banked version of CP/M 3 in BDOS function 10, CP/M 3's console editing input function. Note that typing a character into the last position of the screen, as specified by the Console Width field, must not cause the terminal to advance to the next line.
1B	RO	Console Column Position. This byte contains the current console column position.
1C	RW	Console Page Length. This byte contains the page length, lines per page, of your console. Most systems default this value to 24 lines per page. This default value may be changed by using the GENCPM or the DEVICE utility (see the CPIM Plus (CPIM Version 3) Operating System User's Guide).
1D - 21	RO	Reserved for system use.
22 - 2B	RW	Redirection flags for each of the five logical character devices. If your system's BIOS supports assignment of logical devices to physical devices, you can direct each of the five logical character devices to any combination of up to 12 physical devices. The 16-bit word for each device represents the following: Each bit represents a physical device where bit 15 corresponds to device zero and bit 4 corresponds to device 11. Bits zero through 3 are reserved for system use. You can redirect the input and output logical devices with the DEVICE command (see CP/M Plus (CP/M Version 3) Operating System User's Guide).

Table A-1. (continued)

Offset	RW/RO	Definition
22 - 23	RW	CONIN Redirection Flag.
24 - 25	RW	CONOUT Redirection Flag.
26 - 27	RW	AUXIN Redirection Flag.
28 - 29	RW	AUXOUT Redirection Flag.
2A - 2B	RW	LSTOUT Redirection Flag.
2C	RW	Page Mode. If this byte is set to zero, some CP/M 3 utilities and CCP built-in commands display one page of data at a time; you display the next page by pressing any key. If this byte is not set to zero, the system displays data on the screen without stopping. To stop and start the display, you can press CTRL-S and CTRL-Q, respectively.
2D	RO	Reserved for system use.
2E	RW	Determines if CTRL-H is interpreted as a rub/del character. If this byte is set to 0, then CTRL-H is a backspace character (moves back and deletes). If this byte is set to OFFH, then CTRL-H is a rub/del character, echoes the deleted character.
2F	RW	Determines if rub/del is interpreted as CTRL-H character. If this byte is set to 0, then rub/del echoes the deleted character. If this byte is set to OFF, then rub/del is interpreted as a CTRL-H character (moves back and deletes).
30 - 32	RO	Reserved for system use.
33 - 34	RW	Console Mode. This is a 16-bit system parameter that determines the action of certain BDOS Console I/O functions. (See Section 2.2.1 and BDOS Function 109, Get/Set Console Mode, for a thorough explanation of Console Mode.)

+Table A-1. (continued)

Offset	RW/RO	Definition
35 - 36	RO	Reserved for system use.
37	RW	Output delimiter character. The default output delimiter character is $, but you can change this value by using the BDOS Function 110, Get/Set Output Delimiter.
38	RW	List Output Flag. If this byte is set to 0, console output is not echoed to the list device. If this byte is set to 1 console output is echoed to the list device.
39 - 3B	RO	Reserved for system use.
3C - 3D	RO	Current DMA Address. This address can be set by BDOS Function 26 (Set DMA Address). The CCP initializes this value to 0080H. BDOS Function 13, Reset Disk System, also sets the DMA address to 0080H.
3E	RO	Current Disk. This byte contains the currently selected default disk number. This value ranges from 0-15 corresponding to drives A-P, respectively. BDOS Function 25, Return Current Disk, can be used to determine the current disk value.
3F - 43	RO	Reserved for system use.
44	RO	Current User Number. This byte contains the current user number. This value ranges from 0-15. BDOS Function 32, Set/Get User Code, can change or interrogate the currently active user number.
45 - 49	RO	Reserved for system use.
4A	RW	BDOS Multi-Sector Count. This field is set by BDOS Function 44,, Set Multi-Sector Count.

Table A-1. (continued)

Offset	RW/RO	Defint.tion
4B	RW	BDOS Error Mode. This field is set by BDOS Function 45, Set BDOS Error Mode. If this byte is set to OFFH, the system returns to the current program without displaying any error messages. If it is set to OFEH, the system displays error messages before returning to the current program. Otherwise, the system terminates the program and displays error messages. See description of BDOS Function 45, Set BDOS Error Mode, for discussion of the different error modes.
4C - 4F	RW	Drive Search Chain. The first byte contains the drive number of the first drive in the chain, the second byte contains the drive number of the second drive in the chain, and so on, for up to four bytes. If less than four drives are to be searched, the next byte is set to OFFH to signal the end of the search chain. The drive values range from 0-16, where 0 corresponds to the default drive, while 1-16 corresponds to drives A-P, respectively. The drive search chain can be displayed or set by using the SETDEF utility (see CPIM Plus (Version 3) Operating System User's Guide).
50	RW	Temporary File Drive. This byte contains the drive number of the temporary file drive. The drive number ranges from 0-16, where 0 corresponds to the default drive, while 1-16 corresponds to drives A-P, respectively.
51	RO	Error drive. This byte contains the drive number of the selected drive when the last physical or extended error occurred.
52 - 56	RO	Reserved for system use.

Table A-1. (continued)

Offset	RW/RO	Definition
57	RO	BDOS Flags. Bit 7 applies to banked systems only. If bit 7 is set, then the system displays expanded error messages. The second error line displays the function number and FCB information. (See Section 2.3.13). Bit 6 applies only to nonbanked systems. If bit 6 is set, it indicates that GENCPM has specified single allocation vectors for the system. Otherwise, double allocation vectors have been defined for the system. Function 98, Free Blocks, returns temporarily allocated blocks to free space only if bit 6 is reset.
58 - 59	RW	Date in days in binary since 1 Jan 78.
5A	RW	Hour in BCD (2-digit Binary Coded Decimal).
5B	RW	Minutes in BCD.
5C	RW	Seconds in BCD.
5D - 5E	RO	Common Memory Base Address. This value is zero for nonbanked systems and nonzero for banked systems.
5F - 63	RO	Reserved for system use.

End of Appendix A

Appendix B

PRL File Generation

B. 1 PRL Format

A Page Relocatable Program has an origin offset of 100H bytes that is stored on disk as a file of type PRL. The format is shown in Table B-1.

Table B-1. PRL File Format

Address	Contents
000 1-0002H	Program size
0004-0005H	Minimum buffer requirements (additional memory)
0006-00FFH	Currently unused, reserved for future allocation

0100 + Program size = Start of bit map

The bit map is a string of bits identifying those bytes in the source code that require relocation. There is one byte in the bit map for every 8 bytes of source code. The most significant bit, bit 7, of the first byte of the bit map indicates whether or not the first byte of the source code requires relocation. If the bit is on, it indicates that relocation is required. The next bit, bit 6, of the first byte corresponds to the second byte of the source code, and so forth.

B.2 Generating a PRL

The preferred technique for generating a PRL file is to use the CP/M LINK-80", which can generate a PRL file from a REL relocatable object file. This technique is described in the Programmer's Utilities Guide for The CPIM Family of Operating Systems. A sample link command is shown below.

A>Iink dump[op]

End of Appendix B

Appendix C

SPR Generation

System Page Relocatable, SPR, files are similar in format to PRL files except that SPR files have an origin offset of 0000H (see Appendix B). SPR Files are provided as part of the standard CP/M 3 System: the resident and banked portions of the banked BDOS, named RESBDOS3.SPR and BNKBDOS3.SPR, and the nonbanked BDOS,, named BDOS3.SPR. The customized BIOS must also be generated in SPR format before GENCPM can create a CP/M 3 system. The BIOS SPR file is named BNKBIOS3.SPR for banked systems and BIOS3.SPR for nonbanked systems. A detailed discussion of the generation of BIOS3.SPR or BNK-BIOS3.SPR is provided in the CPIM Plus (CP/M Version 3) Operating System System Guide.

The method of generating an SPR is analogous to that of generating a Page Relocatable Program (described in Appendix B) with the following exceptions:

o If LINK-80 is used, the output file of type SPR is specified with the [os] or [b] option. The [b] option is used when linking BNKBIOS3.SPR.

o Tne code in the SPR is ORGed at 000H rather than 100H.

End of Appendix C

Appendix D

ASCII and Hexadecimal Conversions

This appendix contains tables of the ASCII symbols, including their binary, decimal, and hexadecimal Conversions.

Table D-1. ASCII Symbols

Symbol	Meaning	Symbol	Meaning
ACK	acknowledge	FS	file separator
BEL	bell	GS	group separator
BS	backspace	HT	horizontal tabulation
CAN	cancel	LF	line-feed
CR	carriage return	NAK	negative acknowledge
DC	device control	NUL	null
DEL	delete	RS	record separator
DLE	data link escape	SI	shift in
EM	end of medium	SO	shift out
ENQ	enquiry	SOH	start of heading
EOT	end of transmission	SP	space
ESC	escape	STX	start of text
ETB	end of transmission	SUB	substitute
ETX	end of text	SYN	synchronous idle
FF	form-feed	US	unit separator
		VT	vertical tabulation

Table D-2. ASCII Conversion Table

Binary	Decimal	Hexadecimal	ASCII
0000000	000	00	NUL
0000001	001	01	SOH (CTRL-A)
0000010	002	02	STX (CTRL-B)
0000011	003	03	ETX (CTRL-C)
0000100	004	04	EOT (CTRL-D)
0000101	005	05	ENQ (CTRL-E)
0000110	006	06	ACK (CTRL-F)
0000111	007	07	BEL (CTRL-G)
0001000	008	08	BS (CTRL-H)
0001001	009	09	HT (CTRL-I)
0001010	010	0A	LF (CTRL-J)
0001011	011	0B	VT (CTRL-K)
0001100	012	0C	FF (CTRL-L)
0001101	013	0D	CR (CTRL-M)
0001110	014	0E	SO (CTRL-N)
0001111	015	0F	SI (CTRL-0)
0010000	016	10	DLE (CTRL-P)
0010001	017	11	DC1 (CTRL-Q)
0010010	018	12	DC2 (CTRL-R)
0010011	019	13	DC3 (CTRL-S)
0010100	020	14	DC4 (CTRL-T)
0010101	021	15	NAK (CTRL-U)
0010110	022	16	SYN (CTRL-V)
0010111	023	17	ETB (CTRL-W)
0011000	024	18	CAN (CTRL-X)
0011001	025	19	EM (CTRL-Y)
0011010	026	1A	SUB (CTRL-Z)
0011011	027	IB	ESC (CTRL-[)
0011100	028	1C	FS (CTRL- \)
0011101	029	1D	GS (CTRL-1)
0011110	030	1E	RS (CTRL--)
0011111	031	1F	US (CTRL-_)
0100000	032	20	(SPACE)
0100001	033	21	!
0100010	034	22	'
0100011	035	23	#
0100100	036	24	$

Table D-2. (continued) ersions

Binary	Decimal	Hexadecimal	ASCII
0100101	037	25	%
0100110	038	26	&
0100111	039	27	'
0101000	040	28	(
0101001	041	29	)
0101010	042	2A	*
0101011	043	2B	+
0101100	044	2C	,
0101101	045	2D	-
0101110	046	2E	.
0101111	047	2F	/
0110000	048	30	0
0110001	049	31	1
0110010	050	32	2
0110011	051	33	3
0110100	052	34	4
0110101	053	35	5
0110110	054	36	6
0110111	05S	37	7
0111000	056	38	8
0111001	057	39	9
0111010	058	3A	:
0111011	059	3B	;
0111100	060	3C	<
0111101	061	3D	=
0111110	062	3E	>
0111111	063	3F	?
1000000	064	40	@
1000001	065	41	A
1000010	066	42	B
1000011	067	43	C
1000100	068	44	D
1000101	069	45	E
1000110	070	46	F
1000111	071	47	G
1001000	072	48	H
1001001	073	49	I

Table D-2. (continued)

Binary	Decimal	Hexadecimal	ASCII
1001010	074	4A	J
1001011	075	4B	K
1001100	076	4C	L
1001101	077	4D	M
1001110	078	4E	N
1001111	079	4F	O
1010000	080	50	P
1010001	081	51	Q
1010010	082	52	R
1010011	083	53	S
1010100	084	54	T
1010101	085	55	U
1010110	086	56	V
1010111	087	57	W
1011000	088	58	X
1011001	089	59	Y
1011010	090	SA	Z
1011011	091	5B	[
1011100	092	5C	\
1011101	093	5D	]
1011110	094	5E	^
1011111	095	5F	<
1100000	096	60	'
1100001	097	61	a
1100010	098	62	b
1100011	099	63	c
1100100	100	64	d
1100101	101	65	e
1100110	102	66	f

Table D-2. (continued)

Binary	Decimal	Hexadecimal	ASCII
1100111	103	67	g
1101000	104	68	h
1101001	105	69	i
1101010	106	6A	j
1101011	107	6B	k
1101100	108	6C	I
1101101	109	6D	m
1101110	110	6E	n
1101111	111	6F	o
1110000	112	70	p
1110001	113	71	q
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	|
1111101	125	7D	}
1111110	126	7E	~
1111111	127	7F	DEL

End of Appendix D

Appendix E

BDOS Function Summary

Table E-1. BDOS Function Summary

Function	Function Name	Input Parameters	Returned Values
0	System Reset	none	none
1	Console Input	none	A = Char
2	Console Output	E = char	A = 00H
3	Auxiliary Input	none	A = char
4	Auxiliary Output	E = char	A = 00H
5	List Output	E = char	A = 00H
6	Direct Console I/O	E = 0FFH/ 0FEH/ 0FDH/ char	A = char/status/ none
7	Auxiliary Input Status	none	A = 00/0FFH
8	Auxiliary Output Status	none	A = 00/0FFH
9	Print String	DE = .String	A = 00H
10	Read Console Buffer	DE = .Buffer0	Characters in buffer
11	Get Console Status	none	A = 00/01
12	Return Version Number	none	HL = Version (0031H)
13	Reset Disk System	none	A = 00H
14	Select Disk	E = Disk Number	A = Err Flag
15	Open File	DE = FCB	A = Dir Code
16	Close File	DE = FCB	A = Dir Code
17	Search for First	DE = FCB	A = Dir Code
18	Search for Next	none	A = Dir Code
19	Delete File	DE = FCB	A = Dir Code
20	Read Sequential	DE = FCB	A = Err Code
21	Write Sequential	DE = FCB	A = Err Code
22	Make File	DE = FCB	A = Dir Code
23	Rename File	DE = FCB	A = Dir Code
24	Return Login Vector	none	HL = Login Vector
25	Return Current Disk	none	A = Cur Disk#

Table E-1. (continued)

Function	Function Name	Input Parameters	Returned Values
25	Return Current Disk	none	A = Cur Disk#
26	Set DMA Address	DE = . DMA	A = 00H
27	Get Addr(Alloc)	none	HL = Alloc
28	Write Protect Disk	none	A = 00H
29	Get R/O Vector	none	HL = R/0 Vector
30	Set File Attributes	DE = .FCB	A = Dir Code
31	Get Addr(DPB)	none	HL = DPB
32	Set/Get User Code	E = 0FFH/ user number	A = Curr User/00H
33	Read Random	DE = .FCB	A = Err Code
34	Write Random	DE = .FCB	A = Err Code
35	Compute File Size	DE = .FCB	r0, rl, r2 A = Err Flag
36	Set Random Record	DE = .FCB	r0, rl, r2
37	Reset Drive	DE = Drive Vector	A = 00H
38	Access Drive	none	A = 00H
39	Free Drive	none	A = 00H
40	Write Random with Zero Fill	DE = .FCB	A = Err Code
41	Test and Write Record	DE = FCB	A = 0FFH
42	Lock Record	DE = ..FCB	A = 00H
43	Unlock Record	DE = .FCB	A = 00H
44	Set Multi-sector Count	E = # Sectors	A = Return Code
45	Set BDOS Error Mode	E = BDOS Err Mode	A = 00H
46	Get Disk Free Space	E = Drive number	Number of Free Sectors A = Err Flag
47	Chain to Program	E = Chain Flag	A = 00H
48	Flush Buffers	E = Purge Flag	A = Err Flag
49	Get/Set System Control Block	DE = .SCB PB	A = Returned Byte HL = Returned Word
50	Direct BIOS Calls	DE = .BIOS PB	BIOS Return
59	Load Overlay	DE = .FCB	A = Err Code
60	Call Resident System Extension	DE = .RSX PB	A = Err Code

Note: . indicates the address of

Table E-1. (continued)

Function I	Function Name	Input Parameters	Returned Values
98	Free Blocks	none	A = Err Flag
99	Truncate File	DE = .FCB	A = Dir Code
100	Set Directory Label	DE = .FCB	A = Dir Code
101	Return Directory Label Data	E = Drive	A = Dir label data byte
102	Read File Date Stamps and Password Mode	DE = .FCB	A = Dir Code
103	Write File XFCB	DE = .FCB	A = Dir Code
104	Set Date and Time	DE = .DAT	A = 00H
105	Get Date and Time	DE = .DAT	Date and Time A = seconds
106	Set Default Password	DE = Password	A = 00H
107	Return Serial Number	DE = .Serial field	Serial Number
108	Get/Set Program Return Code	DE = 0FFFFH/ Code	HL Program Ret Code none
109	Get/Set Console Mode	DE = 0FFFFH/ Mode	HL = ConsoleMode none
110	Get/Set Output Delimiter	DE = 0FFFFH/ E = Delimiter	A = OutputDelimiter none
111	Print Block	DE = .CCB	A = 00H
112	List Block	DE = .CCB	A = 00H
152	Parse Filename	DE = .PFCB	See definition

Note: . indicates the address of

End of Appendix E

Index