CPMPPG1.WS4 (= "CP/M-86 Plus Programmers's Guide", section 1)

CP/M-86 Plus -- Programmer's Guide

(Edited by Emmanuel ROCHE.)

Section 1: CP/M-86 Plus system overview

This section introduces the general characteristics of the CP/M-86 Plus operating system for someone wishing to write programs for it. It includes a brief description of the system components, the disk file organization, memory organization, and program and system execution.

System components

The section describes the components of the standard CP/M-86 Plus system. The operating system consists of the Basic Disk Operating System (BDOS), and the customized Basic I/O System (BIOS). Also distributed with the system are the individual command files that contain the utilities of the operating system.

Basic Disk Operating System (BDOS)

The Basic Disk Operating System (BDOS) handles all system calls. Transient programs and the CCP access CP/M-86 Plus facilities by making system calls to the BDOS. Entry to the BDOS is accomplished through the 8086 software interrupt #224, which is reserved by Intel Corporation for use by Digital Research's 8086 Operating Systems. System calls are used to 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 6, "System calls".

Basic Input/Output System (BIOS)

CP/M-86 Plus achieves hardware independence through a clearly defined separation of the hardware-dependent functions, supplied by the BIOS, from the machine-independent primitive functions called in the BDOS. The interface comprises a set of BIOS functions called by the BDOS to perform the hardware-dependent primitives, such as peripheral device input and output. For example, the BDOS calls the console input function of the BIOS to read the next console input character.

The BIOS is tailored to suit a specific computer. However, even when the BIOS primitives are implemented for different designs, the BIOS entry points and

the BDOS remain constant. Therefore, the BDOS and the BIOS modules work together to give CP/M programs hardware-independent access to the CP/M-86 Plus facilities.

Transient programs

A transient program is a file of type CMD. The system loads the transient program from the disk into the available memory for execution.

One transient program can be run in "foreground". A foreground program has access to the keyboard and console screen.

Up to three transient CMD-type programs can run simultaneously with the foreground program. These programs are "background", and do not have access to the keyboard and console screen.

See Section 5, "Program execution", for a more detailed description of background programs.

Resident System Extensions (RSX)

A resident system extension is a program module that can extend or modify one or more operating system functions. RSX modules intercept BDOS system calls, and either perform them, translate them into other system functions, or pass them to the BDOS for normal processing.

It is coded as any other program, except that it requires a special prefix at the beginning of the program. RSX generation and implementation are discussed in Sections 5 and 7.

Memory organization

The 8086 memory architecture varies widely from system to system. Some systems support memory that have non-contiguous adress spaces. In these systems, an application program cannot safely make assumptions about available memory. In the 8086 system, the program must rely on CP/M-86 Plus to keep track of available memory space, and to load programs accordingly.

Physical memory space is defined through a static allocation map located within the BIOS. It is possible to operate CP/M-86 Plus in a memory configuration that is a mixture of up to eight non-contiguous areas of RAM, along with reserved or missing memory regions. In a simple RAM-based system with contiguous memory, the static map defines a single region, usually starting at the end of the BIOS, and extending up to the end of available memory.

The 8086/8088 address space is logically subdivided into segments of up to 64K bytes each, which are allocated to code, data, stack, and extra segments. A

program is categorized as a group of segments that are loaded into memory as a single unit. CP/M-86 Plus incorporates a memory management facility that supports program loading and dynamic memory allocation. It takes advantage of the static relocation inherent in the 8086 processor, and handles modules patterned after 4 Digital Research Memory Models: the 8080 Memory Model, the Small Memory Model, the Compact Memory Model, and the Large Memory Model. Section 5, "Program execution", describes these Memory Models.

Program execution

When CP/M-86 Plus receives a request to load a program from the CCP or another transient program, it checks the program's memory requirements. If sufficient memory is available, the system assigns the required amount of memory to the program, and loads the program. CP/M-86 Plus also performs load-time fixups to address code on data areas larger than 64K. Once loaded, the program can request additional memory from the system for buffer space. When the program is terminated, the system frees both the program memory area and any additional dynamically-allocated memory, unless the program indicates otherwise.

System disk organization

The CP/M-86 Plus system disk organization is standard for CP/M systems. Generally, the first sector(s) contain(s) a bootstrap program used to load CP/M-86 Plus during initial system start. The steps of this procedure are described in the next section. The remainder of the disk contains the disk directories and files, including the generating system image CPMP.SYS.

Initial system load

The initial system load is executed immediately after the computer is turned ON, or when the computer's RESET button is pressed. The initial system load brings CP/M-86 Plus into memory, and gives it control of the computer's resources. The initial system load typically is a multi-stage procedure.

In the first stage, a hardware feature (the ROM-based software associated with system power ON or RESET) loads a small program called the bootstrap program into memory from the system drive (typically drive A).

In the second stage, the CP/M-86 Plus bootstrap program reads the CP/M-86 Plus system file (CPMP.SYS) from the disk. The CPMP.SYS file, which is created by the CP/M-86 Plus system generation utility GENCPM, contains the BDOS and the BIOS system components. Once the bootstrap program has loaded the BDOS and the BIOS into memory, it sends a sign-on message to the console, and passes control to the BDOS.

In the third and final stage of the initial system load, the BDOS initializes the system, and passes control to the command loader. This performs additional

initialization, and loads the CCP. It then transfers control to the CCP. When CCP.CMD receives control, it displays a prompt that references the initial default drive. If a STARTUP.SUB file is present on the default drive, the CCP executes this file (thanks to SUBMIT) before prompting the user for a command line input. See the "CP/M-86 Plus User's Guide" for discussion of submit files. At this point, the initial system load is complete. Note that the system sets the current user number to zero after initial system load.

EOF

CPMPPG2.WS4 (= "CP/M-86 Plus Programmer's Guide", section 2)

(Edited by Emmanuel ROCHE.)

Section 2: Character device I/O

Character 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
PRN: Logical list output device

If your system supports the BIOS DEVTBL function, the CP/M-86 Plus 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 function, then the logical-to-physical device assignments are fixed by the BIOS.

In general, character I/O functions read and write an individual ASCII character, a character string, or a block of characters to and from these devices, or test the device's ready status. For these 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 in a word count field. The maximum value of N in both cases is limited to 64K-1 (0FFFFh). The following list summarizes character 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:
Read a character buffer from AUXIN:
Write a character to AUXOUT:
Write a block of characters to AUXOUT:
Write a character to PRN:
Write a block of characters to PRN:
Interrogate status of CONIN:, AUXIN:, AUXOUT:

The system 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 system call to the logical device performs no action. If a logical input device is not assigned to a physical device, an input system call to the logical device returns a null character (00h). Note that this action depends on your system's BIOS implementation.

Console I/O

Because a program's main interaction with its user is through the console, the system supports many console I/O functions. Console I/O functions can be divided into four categories: basic console I/O, raw console I/O, edited console input, and special console functions.

Basic console I/O

Using the basic console I/O, programs can access the console device for simple input and output. The basic console I/O functions are described in Table 2-1.

Table 2-1. Basic console I/O functions

Mnemonic	Function	Description
C_READ	1	Read a single character from the console
C_STAT	11	Get console status
C_WRITE	2	Write a single character to the console
C_WRITEBLK	111	Write a block of characters to the console
C_WRITESTR	9	Write a string of characters to the console

The input function echoes the character to the console, so that you can identify the typed character. The output functions expand tabs in columns of eight characters.

Control keys control console output, program termination, and duplication of console output on the printer (called "printer echo"). These keys are active when performing basic console functions, but can be selectively disabled using the C_MODE (Function 109) system call, described in Section 6, "System calls". The system recognizes all of the control keys described below when the Console Mode is in the default state.

Ctrl-S stops console output, and Ctrl-Q resumes console output. Ctrl-C terminates the foreground program. Ctrl-P turns on and off printer echo. The system always intercepts Ctrl-S, Ctrl-C, Ctrl-Q, and Ctrl-P characters whenever they are entered at the keyboard, unless they have been specifically disabled by using the C_MODE (Function 109) system call. These characters can appear anywhere in the command line. When a program disables system interception of these characters, the program can use the basic console I/O functions to read these characters.

Raw console I/O

The second category of console I/O is raw console I/O. The C_RAWIO (Function 6) system call can provide raw console I/O in situations where unedited console I/O is required. The C_RAWIO system call actually consists of several

subfunctions that support direct console input, output, and status checks. The system does not filter out special characters during raw console I/O. The raw output subfunction does not expand tabs, and the direct input subfunction does not echo typed characters to the console.

Edited console input

The third category of console I/O accepts edited input from the console. The only function in this category, C_READSTR (Function 10), reads an input line from a buffer, and recognizes certain control characters that edit the input. Table 2-2 lists the editing control characters.

Table 2-2. Line-editing control characters

Char.	System action
Ctrl-A	Moves the cursor one character to the left.
Ctrl-B	Moves the cursor to the beginning of the command line, without any effect on the contents of the line. If the cursor is at the beginning of the line, Ctrl-B moves it to the end of the line.
Ctrl-E	Sends a Carriage Return (Ctrl-M) and a Line Feed (Ctrl-J) to the screen, but does not affect the line buffer.
Ctrl-F	Moves the cursor one character to the right.
Ctrl-G	Deletes the character at the cursor. Has no effect when the cursor is at the end of the line.
Ctrl-H	Deletes the character to the left of the cursor. GENCPM can change this function to the RUB/DEL function.
Ctrl-I	Echoes enough spaces to place the next character position at a tab stop. Tab stops are fixed at every eight character of the physical line.
Ctrl-J	Terminates the input line. The cursor can be positioned anywhere in the line. The entire input line is placed in the input line buffer. Note that the C_READSTR (Function 10) system call does not place a terminating character in the line buffer.
Ctrl-K	Deletes all characters from the cursor position to the end of the line.
Ctrl-M	Terminates the input line. The cursor can be positioned anywhere in the input line. The entire input line is placed in the input line buffer.
Ctrl-R	Retypes the characters to the left of the cursor on a new line.

Ctrl-U Updates the previous line buffer to contain the characters to the left of the cursor. Deletes the current line, and advances to a new line.

Ctrl-W Recalls the previous line if the current line is empty; otherwise, moves cursor to the end of the line.

Ctrl-X Deletes all characters to the left of the cursor.

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.

As an option, the line to be edited can be initialized by the calling program. This option is explained in the C_READSTR (Function 10) system call description in Section 6, "System calls".

C_READSTR also filters input for certain control characters. If you type a Ctrl-C, the system terminates the calling program. C_READSTR 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. In general, C_READSTR accepts as input characters all control characters that it does not recognize as editing control characters. C_READSTR identifies a control character with a leading caret [^] when it echoes the control character to the console. Thus, Ctrl-D appears as ^D in a command line on the screen.

Console Mode and output delimiter

The final category of console I/O functions includes special functions that modify the behavior of other console functions. C_DELIMIT (Function 110) can get or set the current delimiter for the C_WRITESTR (Function 9) system call. The default output delimiter is the dollar sign ("$").

C_MODE (Function 109) gets or sets a 16-bit system variable called the Console Mode. Table 2-3 describes the bits of the console mode variable and their functions (bit 0 is the least significant bit).

Table 2-3. Console Mode variable bits

Bit	Function
0	If this bit is set, C_STAT (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. Bit 0 is usually used in conjunction with bit 3. If bit 3 is zero, a Ctrl-C terminates the program.
1	Setting this bit disables stop and start scroll support for the basic console I/O functions, which comprise the first category of functions described in this section. When this bit is set, C_READ (Function 1)

and C_READSTR (Function 10) read Ctrl-S and Ctrl-Q, and C_STAT (Function 11) returns true if you type these characters. Use this mode in situations where raw console input and edited output is needed. While in this mode, you can use C_RAWIO (Function 6) for input and input status, and C_READ (Function 1), C_WRITESTR (Function 9), and C_WRITEBLK (Function 111) for output without the possibility of the output functions intercepting input Ctrl-S and Ctrl-Q characters.

2 Setting this bit disables tab expansion and printer echo (Ctrl-P) support for C_WRITE (Function 2), C_WRITESTR (Function 9), and C_WRITEBLK (Function 111). Use this mode when non-edited output is required. When this bit is set, C_READ (Function 2) and C_READSTR (Function 10) can read Ctrl-P, and C_STAT (Function 11) returns true when Ctrl-P is entered.

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

All basic console I/O functions are affected by the Console Mode system variable. The Console Mode determines whether Ctrl-S, Ctrl-Q, Ctrl-P, and Ctrl-C characters are recognized or ignored by the system. When the Console Mode is in default mode, the four control characters are recognized. The Console Mode is modified by the program by using the C_MODE (Function 109) system call.

When a transient program begins execution, the Console Mode bits are set to zero, the default state.

Note: The "raw" console I/O functions set bits 1, 2, and 3 in the Console Mode each time they are called. Raw console I/O functions include the S_BIOS (Funtion 50) (when performing console I/O functions) and the C_RAWIO (Function 6) system calls. Subsequent basic console I/O system calls automatically reset these bits. However, if the program uses the C_MODE (Function 109) system call to set the Console Mode bits, then the basic console I/O system calls do not reset the Console Mode bits.

Other character device I/O

For the logical device PRN:, the system provides single character output function L_WRITE (Function 5) and character block output L_WRITEBLK (Function 112).

The system also supports single character output A_WRITE (Function 4) for the logical device AUXOUT:, a single character input A_READ (Function 3) for AUXIN:, a character block input A_READBLK (Function 172) for AUXIN:, a character block output A_WRITEBLK (Function 173) for AUXOUT:, and status functions A_STATIN (Functions 7) and A_STATOUT (Function 8) for AUXIN: and AUXOUT:, respectively.

EOF

CPMPPG3.WS4 (= "CP/M-86 Plus Programmer's Guide", section 3)

(Edited by Emmanuel ROCHE.)

Section 3: CP/M-86 Plus file system

File system overview

CP/M-86 Plus can support up to 16 logical drives. Each logical drives has two regions: a directory area, and a data area. The directory area defines the files that exist on the drive, and identifies the data area space that belongs to each file. The data area contains the file data defined by the directory.

The file system automatically allocates data blocks and directory entries when a program creates or extends a file. The system returns previously allocated space to free space when a program deletes or truncates a file. The allocation and recovery of directory and data space is transparent in the calling program. The system maintains the integrity of the disk files by not allowing the user to ever update the directory area and the data area without using a file system call.

File-access system calls

File-access system calls can be divided into three categories: 1) system calls that operate on directory entries, 2) system calls that operate on records within a file, and 3) miscellaneous system calls that affect the execution of other file-access system calls. System calls in the first group include calls to search for one or more files, rename or truncate a file, set file attributes, assign a password to a file, and compute the size of a file. This group also includes calls to make a new file, open an existing file, and close an existing file.

System calls in the second group include functions to read or write records to a file, either sequentially or randomly, by record position. System read/write calls transfer data in 128-byte units, the basic record size of the file system.

The miscellaneous file-access system calls include calls to set the Current User Number, set the DMA buffer address, parse an ASCII file specification, and set a default password. This group also includes system calls to set the system Multisector Count and the File System Error Mode.

The three groups of file-access system calls all have mnemonics beginning with the F_ prefix, and are listed in Table 3-1.

Table 3-1. File-access system calls

Mnemonic	Function	Description
F_ATTRIB	30	Set file attributes
F_CLOSE	16	Close file
F_DELETE	19	Delete file
F_DMAGET	52	Get DMA base address
F_DMAOFF	26	Set DMA offset
F_DMASEG	51	Set DMA segment
F_ERRMODE	45	Set File System Error Mode
F_MAKE	22	Make a new file
F_MULTISEC	44	Set Multisector Count for file read/write
F_OPEN	15	Open file
F_PARSE	152	Parse filename
F_PASSWD	106	Set default password
F_RANDREC	36	Return record number for file read/write
F_READ	20	Read record sequentially from file
F_READRAND	33	Read record randomly from file
F_RENAME	23	Rename file
F_SFIRST	17	Search for first file entry
F_SIZE	35	Compute file size
F_SNEXT	18	Search for next file entry
F_TIMEDATE	102	Return file time/date stamps, Password Mode
F_TRUNCATE	99	Truncate rest of file
F_USERNUM	32	Set/Get directory user number
F_WRITE	21	Write record sequentially into file
F_WRITERAND	34	Write record randomly into file
F_WRITEXFCB	103	Write file's XFCB
F_WRITEZF	40	Write record randomly with zero fill

Drive-related system calls

The drive-related system calls maintain drive information and drive status. These include calls to select a drive as the default drive, compute a drive's free space, interrogate drive status, and assign a directory label to a drive. The directory label for a drive determine if file passwords are to be used, and the type of date and time stamping to be performed for files on the drive. These system calls all have mnemonics beginning with the DRV_ prefix, as shown in the following table.

Table 3-2. Drive-related system calls

Mnemonic	Function	Description
DRV_ALLOVEC	27	Get drive Allocation Vector
DRV_ALLRESET	13	Reset all drives
DRV_DPB	31	Get Disk Parameter Block address
DRV_FLUSH	48	Flush data buffers
DRV_FREE	39	Free drive
DRV_GET	25	Get default drive
DRV_GETLABEL	101	Get directory label data byte
DRV_LOGINVEC	24	Return Login Vector
DRV_RESET	37	Reset drive

DRV_ROVEC 29 Return drives Read-Only Vector
DRV_SET 14 Set (Select) drive
DRV_SETLABEL 100 Set directory label
DRV_SETRO 28 Set drive to Read-Only
DRV_SPACE 46 Get free space on drive

Disk and drive organization

CP/M-86 Plus can support up to 16 logical drives, identified by the letters A through P 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 for easier file organization. In this manual, references to "drives" mean "logical drives", unless specifically stated otherwise.

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 size of each disk directory is defined within the BIOS. The data area contains the file data defined by the directory. If the drive has adequate storage, a CP/M-86 Plus file can be as large as 32 Megabytes, so that each disk file can contain up to 262,144 (40000h) 128-byte records.

The directory identifies each file with an 8-character filename and a 3-character filetype. Together, these fields must be unique for each file. A file can be assigned an 8-character password, to protect the file from unauthorized access.

Disk file I/O

The maximum file size supported on a drive is 32 Megabytes, so that each disk file can consist of up to 262,144 (40000h) 128-byte records. When a file is created or extended, the system automatically allocates data blocks and directory entries to the file. If no directory or data space is available, the system returns an error to the calling program. Note that any data block allocated to a file is permanently allocated until deletion of the file, or until the system is called to truncate the file. The system does not support any other mechanism for releasing data blocks belonging to a file.

The program can process files sequentially or randomly. The position of each record in a file, called the Random Record Number, identifies its position in the file. For sequentially-created files, the first record's Random Record Number is zero; the last record's Random Record Number is one less than the total number of records in the file. The system can read such a file sequentially beginning at record zero, or at random by record position. You can create a random access file by writing records to the file, and specifying the Random Record Number.

The system automatically allocates data blocks to a file to contain its

records on the basis of the actual record position consumed. When creating a file using random access, you might have logical records in the file to which no data has actually been written. Such files are called "sparse files". A sparse file containing two records, one at position zero, and the other at position 262,143, consumes only two data blocks in the data area. You can only create and access sparse files in random mode.

Under CP/M-86 Plus, 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, the record size on disk is not restricted to 128 bytes. These records are called "physical records". Physical record sizes are a multiple of the basic 128-byte record size. When the physical record size is larger than 128 bytes, the system uses record blocking and deblocking for read or write operations.

Record blocking and deblocking

The physical record size on disk can range from 128 bytes to 4K bytes. This physical record size is normally referred to as the "sector size", which is a multiple of the basic 128-byte unit. As mentioned above, if physical record sizes are larger than 128 bytes, the system uses record blocking and deblocking. 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 records is called "record deblocking". This process is required in read operations.

Record deblocking implies a read-ahead operation. For example, if a program calls the system to read a logical record, the entire physical record is read into a system buffer. The system accesses this buffer for subsequent read operations (assuming, of course, that the requested record resides in the buffer). Conversely, record blocking results in the postponement of physical write operations. For example, if a transient program makes a system call to write a record, the logical record is placed in a system buffer, and is not written to the disk until the buffer is needed in another I/O operation, or it fills up. The program can force a disk write by using the DRV_FLUSH (Function 48) system call. The system automatically makes a DRV_FLUSH system call when called to close a file. Thus, it is sufficient to close a file to ensure that all pending physical buffers for that file are written to the disk.

Multisector I/O

To increase the speed of sequential file access, the system has a feature called "multisector I/O", which provides the capability of reading or writing multiple 128-byte records in a single system call. In a multisector I/O operation, the file system bypasses, when possible, all intermediate record buffering. Data is transferred directly from a drive to user memory, and vice versa.

The number of records that can be supported with multisector I/O ranges from 1

to 128. This value is called the "Multisector Count", and can be set by using the F_MULTISEC (Function 44) system call. The default value for the Multisector Count is one. Note that the greatest potential performance increases are obtained when the Multisector Count is set to 128. In this case, however, the program must provide a 16K buffer.

DMA buffer

DMA is an acronym for Direct Memory Access, a term used for file I/O operations which transfer file records directly to memory, and vice versa. Under CP/M-86 Plus, the current DMA buffer is usually defined as the buffer in memory where a record resides before a disk write, and after a disk read. If the Multisector Count is one, the size of the buffer is 128 bytes. However, if the Multisector Count is greater than one, then the size of the DMA buffer must equal n*128, where n equals the Multisector Count.

The system defaults the DMA buffer offset to 0080h, and the DMA buffer segment or base to the initial Data Segment of the program. You can use the F_DMASEG (Function 51) and/or F_DMAOFF (Function 26) system calls to change the location of the DMA buffer.

Although the DMA buffer is normally used for file record transfer, it is sometimes used for other purpose. For example, system calls that check and assign file passwords require that the file password be placed in the first eight bytes of the DMA buffer before issuing a file-access system call. As another example, the DRV_SPACE (Function 46) system call returns its results in the first 3 bytes of the current DMA buffer. When the DMA buffer is used in this context, the size of the buffer in memory is determined by the specific requirements of the system call.

File byte counts

Although the logical record size of CP/M-86 Plus is 128 bytes, the file system does provide a mechanism for the user to store and retrieve the byte count for the last record of a file. The F_ATTRIB (Function 30) system call can set the last record byte count. Conversely, the F_OPEN (Function 15), F_SFIRST (Function 17), and F_SNEXT (Function 18) system calls return the byte count for the last record of a file (see Table 3-3, field CS of the FCB).

The F_OPEN (Function 15) system call will return the byte count in the CR (current record) field only if CR = 0FFh before the F_OPEN is performed. The F_SFIRST (Function 17) and F_SNEXT (Function 18) system calls always return the byte count in the CS field of the FCB. This field is reserved during all other operations.

Note that the file system does not accept or update the byte count value during read or write operations. It is the programmer's responsibility to maintain this value. But the F_MAKE (Function 22) system call does set the byte count value to zero when it creates a file entry in the directory.

File Control Block (FCB)

Each file being accessed through the system has a corresponding File Control Block (FCB), which provides information needed for all file operations. The FCB is a 33 to 36 byte area in the transient program's memory space that is set up for each file. The FCB serves as an important channel for information exchange between a program and file-access system calls. A program initializes an FCB to specify the drive location, filename and filetype fields, and other information required to make a file-access system call. For example, in an F_OPEN (Function 15) system call, the FCB specifies the name and location of a file to be opened. In addition, the file system uses the FCB to maintain the current state and record position of an open file. Some file-access system calls use special fields within the FCB for invoking options. Other file-access system calls use the FCB to return data to the calling program. All random I/O system calls require the calling program to specify the Random Record Number in a 3-byte field at the end of the FCB.

In order to perform file-access operations on any file, an FCB must first be "activated" for the file. This process of activating a file requires the calling program to either open the file by using the F_OPEN (Function 15) system call for an existing file, or the F_MAKE (Function 22) system call for creating a new file. Be sure to verify, by checking the return code of a system call, that the open or make operation is successful. After a file's FCB has been activated, the program can modify only certain fields of the FCB. These fields are marked with an "(*)" in Table 3-3.

When making a file-access system call, the program passes an FCB address to the system. This address has two 16-bit components: register DX containing the offset, and register DS containing the segment. The length of the FCB data area depends on the system call. For most system calls, the minimum length is 33 bytes. For other system calls, the minimum FCB length is 36 bytes.

FCB format and structure

Figure 3-1 describes the FCB format:

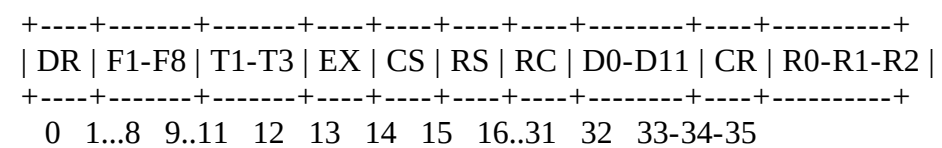


Figure 3-1. File Control Block (FCB) format

Table 3-3 describes the FCB structure.

Table 3-3. File Control Block (FCB) structure

Format: Byte Offset -- Field Name
Description

0 -- DR
Drive code (0-16).
0 --> Use default drive for file
1 --> Auto disk select drive A
2 --> Auto disk select drive B
...
16 --> Auto disk select drive P

1..8 -- F1-F8
Contain the filename in ASCII uppercase, with high bit = 0. F1'-F8' denote the high-order bit of these positions, and are called "file attribute bits". F1' through F4' are reserved attributes for program (CMD) files. Otherwise, F1' through F4' are available for user definition. F5' and F6' are referred to as "interface attributes", and are used as option flags for the following system calls: F_CLOSE (Function 16), F_DELETE (Function 19), F_MAKE (Function 22), and F_ATTRIB (Function 30). The F5' and F6' values are discussed in the corresponding system call descriptions in Section 6, "System calls". F7' and F8' are reserved for system use.

9..11 -- T1-T3
T1, T2, T3 contain the filetype in ASCII uppercase, with high bit = 0.
T1', T2', T3' denote the high bit of these positions, and are file attribute bits.
T1' = 1 --> Read-Only (RO) file
T2' = 1 --> System (SYS) file
T3' = 1 --> Archived file
These file attributes are described later in this section.

12 -- EX
Contains the current extent number. This is set to 0 by the calling program, but it can range from 0 to 31 during file I/O. Each extent contains 128 logical record.

13 -- CS
File byte count after F_OPEN, F_SFIRST or F_SNEXT.

14 -- RS
Reserved for system use.

15 -- RC
Record count for extent EX. This field takes on value from 0 to 255 (values greater than 128 imply a record count of 128).

16-31 -- D0-D11
Allocation field normally filled in by the system, and reserved for system use. Also used by the F_RENAME (Function 23) and F_PARSE (Function 152) system calls.

32 -- CR (*)
Current record to read or write in a sequential file operation. This field is normally set to zero by the calling program when a file is opened or created.
File byte count after F_OPEN with CR = 0FFh.

33-34-35 -- R0,R1,R2 (*)

Optional Random Record Number in the range 0 to 262,143 (0 to 3FFFFh). R0, R1, R2 constitutes an 18-bit value with low byte R0, middle byte R1, and high byte R2.

Note: Fields designated by "(*)" (CR and R0,R1,R2) are fields that can be modified by the program after the FCB has been activated.

FCB initialization and usage

The calling program must initialize the DR, F1-F8 and T1-T3 fields of the FCB before making the following file-access system calls:

F_ATTRIB	Set file attributes
F_DELETE	Delete file
F_MAKE	Make a new file
F_OPEN	Open file
F_RENAME	Rename file
F_SFIRST	Search for first file entry
F_SIZE	Compute file size
F_SNEXT	Search for next file entry
F_TIMEDATE	Return file time/date stamps, Password Mode
F_TRUNCATE	Truncate rest of file
F_WRITEXFCB	Write file's XFCB

The calling program must also set the EX field of the FCB before making F_MAKE, F_OPEN, F_SFIRST, and F_WRITEXFCB (Functions 22, 15, 17, and 103) system calls. Set this field to zero, except for the F_WRITEXFCB (Function 103) system call.

To use the F_RENAME (Function 23) system call, the calling program must place the new filename and filetype in bytes D1 through D11.

The remaining file-access calls that use FCBs require an FCB that has been initialized by a prior file-access system call. For example, the F_SNEXT (Function 18) system call expects an FCB initialized by a prior F_SFIRST (Function 17) system call. In addition, the F_READ, F_READRAND, F_WRITE, F_WRITERAND, and F_WRITEZF (Functions 20, 33, 21, 34, and 40) system calls require an FCB that has been activated for record operation. Under CP/M-86 Plus, only the F_OPEN and F_MAKE (Functions 15 and 22) system calls can activate an FCB.

If the program is to process a file sequentially from the beginning of the file, the CR (current record) field must be set to zero before making the first read or write call. When processing a file in random mode, bytes 33 through 35 of the FCB must be set to the requested Random Record Number. The F_TRUNCATE (Function 99) system call also requires the random record field to be initialized.

The F_SFIRST, F_SNEXT, and the F_DELETE (Functions 17, 18, and 19) system calls support multiple or ambiguous file references. In general, a question mark ("?") in the filename, filetype, or EX fields matches all values in the corresponding positions of the directory entries during a directory search

operation. File directory entries (henceforth referred to as "Directory FCBs") maintained in the directory area of each disk drive have the same format as FCBs, except for byte 0 which contains the file's user number, and bytes 32 through 35, which are excluded. The F_SFIRST and F_SNEXT (Functions 17 and 18) system calls also recognize a question mark in the FCB DR field and, if specified, return all directory entries on the disk, regardless of user number, including empty entries.

The system updates the memory copy of the Directory FCB during file processing, to maintain the current position within the file. During file write operations, the system also updates the memory copy of the Directory FCB, to record the allocation of data blocks to the file. At the termination of file processing, the F_CLOSE (Function 16) system call permanently records this information in the directory.

Note that the system does not necessarily record the data blocks allocated to a file during write operations in the disk directory until the calling program issues an F_CLOSE (Function 16) system call. Therefore, a program that creates or modifies files must close the files at the end of processing. Otherwise, data might be lost.

File naming conventions

The "CP/M-86 Plus User's Guide" discusses details of the file naming conventions for CP/M. In general, however, a file specification consists of four parts: the drive specifier (d:), the filename, the filetype, and the file password. A command line file specification takes the general form:

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

The drive specifier names the drive on which the file is located. An eight-character filename and a three-character filetype field identify each file in a directory. Programs can also assign an eight-character password to a file, to protect it from unauthorized access. All system calls that involve file operations specify the requested file by filename and filetype. Multiple files can be specified by a wildcard file specification. An ambiguous file specification uses one or more question marks ("?") in the filename or filetype to indicate that any character matches that position. Thus, a filename and filetype consisting of all questions marks (equivalent to a command line specification of "*.*") matches all files in the directory that belong to the Current User Number.

A program can use the F_PARSE (Function 152) system call to translate (parse) the file specification, and to place the required FCB fields in a user-defined FCB address.

File passwords

CP/M-86 Plus provides file password protection on CP/M disks in one of three modes. Table 3-4 shows the difference in access level allowed to system calls

when the password is not supplied for a password-protected file.

Table 3-4. Password protection modes

Mode	Access level allowed without password
1) Read	Cannot be read, modified, or deleted.
2) Write	Can be read, but not modified or deleted.
3) Delete	Can be read and modified, but not deleted.

If a file is password-protected in Read mode, a program must supply the password to open the file. Programs cannot write to a file protected in Write mode without the password. A file protected in Delete mode allows read and write access, but a program must specify the password to delete the file, rename the file, or 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.

The F_MAKE (Function 22) or the F_WRITEXFCB (Function 103) system call can be used to assign a password to a file. The F_WRITEXFCB system call can also be used to change a password, if the original password is supplied.

The following system calls test for passwords for password-protected files:

DRV_SETLABEL
F_ATTRIB
F_DELETE
F_OPEN
F_RENAME
F_TRUNCATE
F_WRITEXFCB

You do not need to continually supply the password to access your files if you establish a default password before you access a file that requires a password. You can establish a default password by using the F_PASSWD (Function 106) system call. The operator can also establish the default password prior to program execution by using the SET utility. See the "CP/M-86 Plus User's Guide".

The system uses the default password for password-protected file if you do not supply the password in the system call. This password is in effect until replaced by another F_PASSWD (Function 106) system call.

File passwords are eight characters long. To make a system call that requires a password, the program must place the password in the first eight bytes of the current DMA buffer, or establish a default password.

File date and time stamps

CP/M-86 Plus supports three kinds of file stamping: create, access, and update. Create stamps record when the file was created, acces 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.

Date and time stamps are supported on CP/M media only if the directory of the disk has been properly initialized. The INITDIR utility initializes a directory for date and time stamping. CP/M disk's time and date stamping is not performed if the CP/M disk's directory label is absent, or does not specify time and date stamping, or if the drive is Read-Only.

Note that the CP/M directory label is also time-stamped. Time stamp fields in the last eight bytes of the directory label show when it was created and last updated. Access stamping is not supported for directory labels.

The file system uses system date and time when it records a date and time stamp. The operator can set the system time and date by using the DATE command (see the "CP/M-86 Plus User's Guide"). A program can also use the T_SET (Function 104) system call to set time and date.

File attributes

The high-order bits of the FCB filename F1' through F4' and filetype T1' through T3' are called "attribute bits". The program can assign or interrogate the following attributes:

T1' = 1 --> Read-Only (RO) attribute
T2' = 1 --> System (SYS) attribute
T3' = 1 --> Archive attribute

If a file is set to Read-Only, the system cannot write to the file.

If the file has the SYS attribute, the system treats it as a system file. For example, certain utilities such as DIR and PIP do not include it unless explicitly specified. Also, the system allows other users to access user zero SYS files on a Read-Only basis.

When a file has the Archive attribute, it indicates that the file has not been modified since a previous archive (back-up) function. An archive program should interrogate this attribute, and set the Archive attribute when it copies a file. The system automatically resets the Archive attribute to off when the file is updated. The PIP utility supports file archiving.

The file attribute bits F1' through F4' and T1', T2', and T3' are recorded in the file's Directory FCB. These attributes can be set or reset only by the F_ATTRIB (Function 30) system call.

User number conventions

The CP/M-86 Plus file system subdivides the directory area into sixteen logically independent directories. These directories are identified by user

number 0 through 15. Physically, all user directories share the directory area of a drive. However, files existing under different user numbers are independent of each other; for example, files with the same name but different user numbers can reside on the same drive with no conflict.

During system operation, CP/M-86 Plus runs with the user number set to a single value. This value is referred to as the "Current User Number". The system operator can change the Current User Number at the console by using the USER built-in command (see the "CP/M-86 Plus User's Guide"). A transient program can change the Current User Number by using the F_USERNUM (Function 32) system call.

Only one user number is active in the system at any time. All file and directory operations reference only directory entries associated with this Current User Number, except for user zero SYS (System) files. Under CP/M-86 Plus, files under user zero marked with the SYS attribute can be read by other users. This convention allows utilities, including overlays and any other commonly accessed files, to reside on user zero but remain available to other users. This eliminates the need to copy commonly used utilities to all user numbers on a directory.

A program can access files on different user numbers by setting the user number to the appropriate user number with the F_USERNUM (Function 32) system call. However, an error occurs if a program attempts to read from or write to a file opened under a different user number.

Directory entries

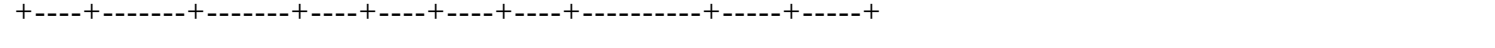
Directory entries are 32-byte structures created and updated in the directory area by the system. They represent a permanent, static recording on disk of the file's name, attributes, date and time stamps, password, and associated data blocks. Directory entries are not processed directly by users; that is to say: only the system can update the directory area of the drive.

There are four types of directory entries recognized and supported by the system:

1. A Directory Label entry
2. File directory entry (Directory FCB)
3. Password directory entry (XFCB)
4. Date and Time Stamp directory entry (SFCB)

Directory label

A directory label can be included in a CP/M drive's directory. The directory label specifies if the system is to support file passwords, and if the system is to maintain date and time stamping for files on the drive. Figure 3-2 shows the directory label format.



```
| DR | Fname | Ftype | DL | S1 | S2 | RC | Password | TS1 | TS2 |
+----+-------+-------+----+----+----+----+----------+-----+-----+
  0   1...8   9..11   12   13   14   15    16..23     24    28
```

Figure 3-2. Directory Label format

Table 3-4 describes the structure and contents of the CP/M directory label.

Table 3-4. CP/M Directory Label structure

Format: Byte Offset -- Field Name
Description

0 -- DR
Directory label indicator.

1..8 -- Fname
Directory label name.

9..11 -- Ftype
Directory label type.

12 -- 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

13,14,15 -- S1,S2,RC
Reserved for future use.

16-23 -- Password
8-byte password field (encrypted).

24-27 -- TS1
4-byte creation/access time stamp field.

28-31 -- TS2
4-byte update stamp field.

Only one directory label can exist in a drive's directory area. The directory label name and type fields are not used to search for a directory label; they are used to identify the drive.

The DRV_SETLABEL (Function 100) system call can be used to create a CP/M directory label. This function can also be used to assign a directory label password. This password, if assigned, cannot be circumvented, whereas file password protection on a drive is an option controlled by the directory label.

The file system provides no specific system call to read the directory label directly. However, a program can read the directory label data byte by using the DRV_GETLABEL (Function 101) system call. A program can also use the

F_SFIRST and F_SNEXT (Functions 17 and 18) system calls to find a directory label. The directory label is identified by a value of 32 (20h) in the DR field (byte 0) of the directory label structure.

File directory entry (Directory FCB)

Each file on the CP/M drive has at least one 32-byte entry in the directory. This entry contains the filename, filetype, and other data, plus 16 bytes that define the allocation blocks assigned to the file. Figure 3-3 shows the Directory FCB format.

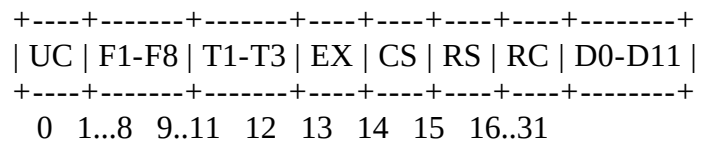


Figure 3-3. CP/M Directory FCB format

Table 3-5 describes the Directory FCB structure.

Table 3-5. Directory FCB structure

Format: Byte Offset -- Field Name
Description

0 -- UC
User code (0-15).

1..8 -- F1-F8
Contain the filename in ASCII uppercase, with high bit = 0. F1'-F8' denote the high-order bit of these positions, and are the file and interface bits described in Section 3.

9..11 -- T1-T3
Contain the filetype in ASCII uppercase, with high bit = 0. T1', T2', T3' denote the high bit of these positions, and are the file attribute bits described in Section 3.
T1' = 1 --> Read-Only (RO) file
T2' = 1 --> System (SYS) file
T3' = 1 --> Archived file

12 -- EX
Extent number for the file (0-31).

13 -- CS
File byte count for last record.

14 -- RS
Reserved for system use.

15 -- RC
Record count for extent EX. This field takes on value from 0 to 255 (values

greater than 128 imply a record count of 128).

16-31 -- D0-D11
Allocation blocks assigned to this extent.

Password directory entry (XFCB)

The CP/M-86 Plus file system uses extended directory entries called "XFCBs" for file passwords. Figure 3-4 shows the XFCB format.

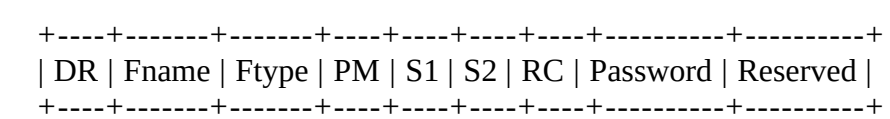


Figure 3-4. CP/M XFCB format

Table 3-6 describes the structure of this extended directory entry.

Table 3-6. Password directory entry (XFCB) structure

Format: Byte Offset -- Field Name
Description

0 -- DR
XFCB indicator (used by the system only).

1..8 -- Fname
Filename.

9..11 -- Ftype
Filetype.

12 -- PM
Password Mode.
Bit 7: Read mode
Bit 6: Write mode
Bit 5: Delete mode

13-14-15 -- S1,S2,RC
Reserved for system use.

16-23 -- Password
8-byte password field (encrypted).

24-31 -- Reserved
8-byte area reserved for future use.

An XFCB can only be created on a drive that has a directory label, and only if the directory label enables password protection. For drives in this state, there are two ways to create an XFCB for file: with the F_MAKE (Function 22) system call, or with the F_WRITEXFCB (Function 103) system call. The F_MAKE

(Function 22) system call creates an XFCB if the calling program requests that a password be assigned to the created file. The F_WRITEXFCB (Function 103) system call creates an XFCB when it is called to assign a password to an existing file.

Date and time stamps (SFCBs)

The CP/M-86 Plus 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 directory entry in the directory. Each SFCB maintains the date and time stamps for the previous three directory entries, as is shown in Figure 3-5.

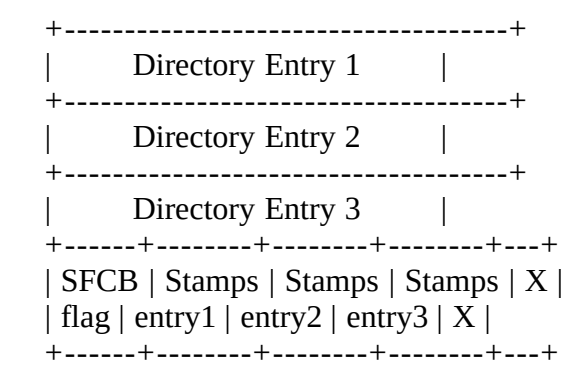


Figure 3-5. Directory record with SFCB

This figure shows a 128-byte directory record containing an SFCB. Directory records have four directory entries, each 32 bytes long; SFCBs always occupy the last 32-byte position in the directory record.

The SFCB itself contains five fields. The first field is a single byte containing the value 33 (21h); this field identifies the SFCB within the directory. The next three fields, called the SFCB subfields, are each 10 bytes in length, and contain the date and time stamps for their corresponding FCB entries in the directory record. The last byte of the SFCB is reserved for system use. Figure 3-6 shows the detail of the SFCB subfields.

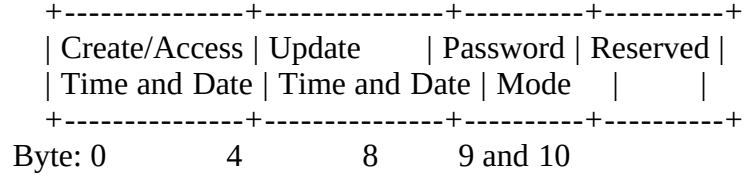


Figure 3-6. SFCB subfields

An SFCB subfield only contains valid information if its corresponding entry in the directory record is the file's first directory FCB. For password-protected files, the SFCB subfield also contains the Password Mode of the file; this subfield is zero for files without password protection. The F_SFIRST and F_SNEXT (Functions 17 and 18) system calls directly access SFCBs. Also, the F_TIMEDATE (Function 102) system call returns the file date and time stamps, and the Password Mode of the file.

Drive status

After initial system load, all drives are initialized to the reset state. This means that their directories have not been read into the system, or that the drives are not logged in. As a drive is referenced, the drive is automatically logged in; that is to say, its directory is brought into the system for file operations. The drive's Allocation Vector is initialized. The Allocation Vector records the allocation and de-allocation of data blocks to files. This vector is updated as files are created, extended, deleted, or truncated. Another system function performed during drive log-in is the initialization of the directory checksum vector. The system uses the directory checksum vector to detect media changes on a drive. The drive remains in logged-in state until reset by using the DRV_RESET (Function 37) or DRV_ALLRESET (Function 13) system call.

The primary use of the drive reset functions is to prepare for a media change on a drive. Resetting a drive has two important 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. Be sure to close your files prior to resetting a drive. Although the system automatically logs in removable media when media changes are detected, you should still explicitly reset a drive before prompting the user to change disks.

File System Error Mode

The file system handles the majority of errors it detects by setting return codes in registers AL and AH, and returning to the calling program. Two examples of this kind of error are the "file not found" error for the F_OPEN (Function 15) system call, or the "reading unwritten data" error for the F_READRAND (Function 33) system call. More serious errors, such as disk I/O errors, are separated in a different category called "physical and extended errors". The way the system responds upon encountering these types of errors depends on the File System Error Mode.

The File System Error Mode determines how the file system responds to physical or extended error conditions, and can be one of the following:

- "Return" error mode
  The system returns to the calling program with return codes in registers AL and AH.

- "Display and Terminate" error mode (Default Mode)
  The system displays the error message, and terminates the calling program.

- "Display and Return" error mode
  The system displays the error message, and returns to the calling program.

In the default state, the system displays the error message, and terminates the calling program. In "Return" error mode, the system returns control to the calling process with the error codes in registers AL and AH. In "Display and Return" error mode, the system displays the error message, and sets the error codes in registers AL and AH before returning to the calling program.

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
```

The "d" identifies the drive selected when the error condition is detected; "error message" identifies the error; "nn" is the system function code, and "filename.typ" identifies the file (if any) affected by the error.

The File System Error Mode can be changed by using the F_ERRMODE (Function 45) system call.

EOF

CPMPP4.WS4 (= "CP/M Plus Programmer's Guide", section 4)

(Edited by Emmanuel ROCHE.)

Section 4: System environment

The operating system environment is determined by the status of the drives in the system, the current memory allocation state, and by certain system flags and data. This information is necessary for the program to define its own execution environment.

When the system is first powered up, the operating system is brought into memory with certain default parameters. For instance, the current user number is set to zero, and the default drive to the initial default drive defined at system generation. Console or terminal characteristics, such as console width and page length are defined. The Console Page Mode and the Console Mode are set to default values. Certain buffer addresses are defaulted to certain values. The date and time can also be initialized. The File System Error Mode is set to its default value. The Drive Search Chain is set to the default drive. All these system parameters can be changed by system utilities (see the "CP/M-86 Plus User's Guide"), or by system calls.

System date and time

The system date and time is initialized to midnight on 1 January 1980, unless the BDOS sets the date and time from a real-time clock. The DATE and TIME commands (see the "CP/M-86 Plus User's Guide") or the T_SET, T_GET, or S_SYSVAR (Functions 104, 105, or 49) system calls can be used to set or get the date and time.

Console characteristics

The system is also defined with default console characteristics. Some of these characteristics can be displayed and modified by using the DEVICE command (see the "CP/M-86 Plus User's Guide"). Others are accessed by system calls. The following console characteristics can be modified by DEVICE:

- Console width
- Console page length

The "console width" and the "console page length" define the size of the screen. The "console width" is the maximum number of columns, and the "console page length" is the maximum number of lines to be displayed on the screen. The DEVICE command (see the "CP/M-86 Plus User's Guide") can be used to redefine the console size.

- Console Page mode

The "console page mode" tells the system whether CP/M-86 Plus utilities are to display information on the screen a page (as defined by the "console page length") at a time, or whether to display information continuously by scrolling (to be interrupted by Ctrl-S, and continued by Ctrl-Q). If the "console page mode" is ON, then the system displays information on the screen a page at a time. The "console page mode" can be turned OFF or ON by using the SETDEF command (see the "CP/M-86 Plus User's Guide"), or by using the S_SYSVAR (Function 49) system call.

The Console Mode and Output Delimiter are set by the C_MODE and C_DELIMIT (Functions 109 and 110) system calls, as discussed in Section 2.

File system variables

Some of the file-access system calls are affected by certain file system variables. These are described in previous sections of this guide, and include the following:

- Current DMA Buffer address (F_DMAGET, F_DMAOFF, F_DMASEG)
- Current Disk (DRV_SET, DRV_GET)
- Current User Number (F_USERNUM)
- Default Password (F_PASSWD)
- Multisector Count (F_MULTISEC)
- File System Error Mode (F_ERRMODE)

All of the above system variables have corresponding system calls that can be used to set or interrogate the system variables.

The S_SYSVAR system call

The S_SYSVAR (Function 49) system call allows access to system variables that do not have specific system calls associated with them. The S_SYSVAR (Function 49) system call (see Section 6, "System calls") describes the details of accessing or updating these system variables. They include the following:

- Console Width
- Console Page Length
- Console Page Mode
- System ticks per second
- Temporary Drive
- Date and Time

The console-related variables have already been discussed above. The "system ticks per second" can be used by an application program which requires a real-time system environment. The "date and time" variables are included to allow the program to also set the seconds field of the time.

The "temporary drive" is the drive used by the system each time it creates

temporary files (with filetype $$$). Of course, the faster the drive, the better the performance of the system so, if the computer has a RAMdisk, it should be used for that purpose. (If the RAMdisk is big, you can also use it as your work disk, instead of the floppies or hard disks (which could "sleep" after one minute of inactivity), but don't forget to save the result of your work before turning OFF the computer!)

EOF

CPMPP5.WS4 (= "CP/M-86 Plus Programmer's Guide", section 5)

(Edited by Emmanuel ROCHE.)

Section 5: Program execution

A CP/M transient program is a file of type CMD (denoting a ComManD file) that is loaded from disk, and normally resides in memory only during its operation. You can initiate a transient program by entering the program name at the system console.

The steps required to generate and execute a transient program under the CP/M-86 Plus operating system are the following:

1. Code the program using one of the Memory Models described below.

2. Assemble or compile the source code.

3. Generate a command file (type CMD) using the Digital Research's LINK-86 utility described in the "Programmer's Utilities Guide for the CP/M-86 Family of Operating Systems".

4. Load and execute the program via the CCP, by typing the program name at the system console.

Table 5-1 defines certain 8086 terms.

Table 5-1. 8086 terms

Term	Meaning
Nibble	4-bit value
Byte	8-bit value
Word	16-bit value
Double Word	2 contiguous words
Paragraph	16 contiguous bytes
Paragraph Boundary/Address or Segment Address:	An address divisible evenly by 16 (low order nibble 0)
Segment	Up to 64K contiguous bytes addressable in a paragraph boundary
Segment Register	One of the CS, DS, ES, or SS segment
Offset	16-bit displacement relative to a segment register
Group	A segment-register-relative relocatable program unit
Address	The effective 20-bit memory address derived from the composition of a segment register value with an offset value. This 20-bit value is equal to the segment value times 10h plus the offset value.

Memory Models

CP/M-86 Plus supports four types of Memory Models: the 8080 memory model, the Small memory model, the Compact memory model, and the Large memory model. When the system loads a program, it initializes the segment registers (CS, DS, ES, and SS), the instruction pointer (IP), and the stack pointer (SP). These values are determined by the specific type of memory used by the transient program. The system also initializes certain fields in the Base Page area (described in detail later in this section). Table 5-2 summarizes the four Memory Models.

Table 5-2. CP/M-86 Plus Memory Models

Model	Group relationships
8080	Code and Data groups overlap
Small	Independent Code and Data groups
Compact	Independent Code, multiple Data groups
Large	Multiple Code and Data groups

The 8080 Memory Model supports programs that are directly translated from an 8080 environment, where code and data are intermixed. The 8080 Memory Model consists of one group that contain all the code, data, and stack areas. Segment registers are initialized to the starting address of the region containing this group. The segment registers can, however, be managed by the application program during execution, so that multiple segments in the code group (which is larger than 64K byte) can be addressed.

The Small Memory Model is similar to that defined by Intel, where the program consists of an independent code group and a data group. The code and data groups often consist of, but are not restricted to, single 64K byte segments.

The Compact Memory Model is used when any of the extra, stack, or auxiliary groups are present in program. Each group can consist of one or more segments, but if any group exceeds one segment in size, or if auxiliary groups are present, then the application program must manage its own segment registers during execution, in order to address all code and data areas.

The Large Memory Model supports programs with multiple segments per group. Code, data, stack, and extra groups may be up to 1 Megabyte in size.

These four Memory Models differ primarily in how the operating system initializes the segment registers when it loads a transient program. The system determines the memory model used by a transient program by examining the Header Record of the CMD file.

For all models, the operating system initializes an internal 96-byte stack area. The first two words of this stack are reserved for the double word return for termination by a RETF (Far Return) instruction. Figure 5-1 shows the initial program stack for all models.

```
+-------------+
| Ret Segment |
+-------------+
| Ret Offset  | <-- Far Return address SS:SP
```

Figure 5-1. Initial program stack

The transient program can terminate by using the P_TERMCPM (Function 0) or P_TERM (Function 143) system call, or by executing a RETF (Far Return) instruction when the SS and SP still point to the initial program stack.

8080 Memory Model

The 8080 Memory Model is assumed when the transient program Header Record contains only a code group. In this case, the system initializes the CS, DS, and ES registers to the beginning of the code group, and sets the SS and SP registers to the 96-byte initial stack area. The system sets the Instruction Pointer (IP) register to 0100h. This allows the Base Page to the beginning of the code group.

Following program load, the 8080 Memory Model appears as in Figure 5-2. The intermixed code and data areas are indistinguishable.

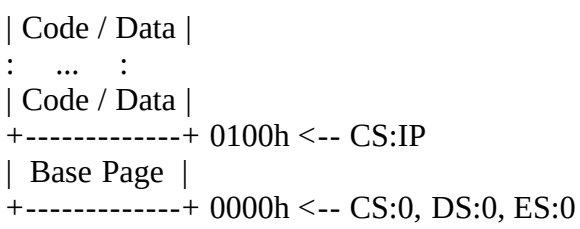
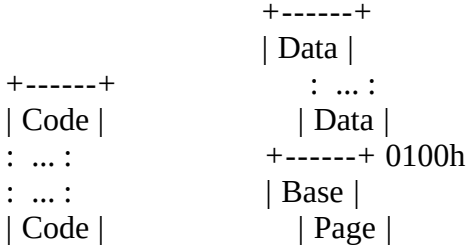


Figure 5-2. 8080 Memory Model

Small Memory Model

The Small Memory Model is used when the transient program contains both a code and data group. (In RASM-86, all code is generated following a CSEG directive. Data is defined following a DSEG directive, with the origin of the Data Segment independent of the Code Segment.) In this memory model, the system sets the CS register to the beginning of the code group, the IP to 0000h, the DS and ES registers to the beginning of the data group, and the SS and SP registers to the 96-byte initial stack area. Following program load, the Small Memory Model appears as shown in Figure 5-3.



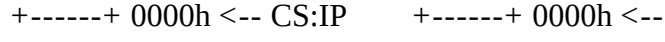


Figure 5-3. Small Memory Model

The machine code begins at CS + 0000h, the Base Page values begin at DS + 0000h, and the data area begins at DS + 0100h.

Compact Memory Model

The Compact Memory Model is used when code and data groups are present, along with one or more of the remaining stack, extra, or auxiliary groups. In this case, the system sets the CS, DS, and ES registers to the base addresses of their respective areas, with the IP set to 0000h, and the SS and SP registers set to the 96-byte initial stack area.

Figure 5-4 shows the initial configuration of the segments in the Compact Memory Model. The values of the various segment registers can be changed during execution by loading from the initial values placed by the system in the Base Page. This allows access to the entire memory space.

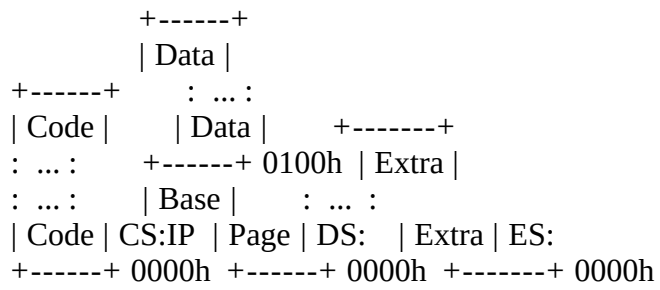


Figure 5-4. Compact Memory Model

If the assembly language transient program intends to use the stack group as a stack area, the SS and SP registers must be set upon entry. The SS and SP registers remain in the initial stack area, even if a stack group is defined.

Although it appears that the SS and SP registers should be set to address the stack group, there are two contradictions. First, the assembly language transient program might be using the stack group as a data area. In that case, the stack values set by the system to allow a Far Return to terminate a transient program could overwrite data in the stack area.

Second, the SS register would logically be set to the base of the group, while the SP would be set to the offset of the end of the group. However, if the stack group exceeds 64K, the address range from the base to the end of the group exceeds a 16-bit offset value.

Large Memory Model

CP/M-86 Plus supports programs with multiple segments per group, with up to 1 Megabyte per group (code, data, stack, extra).

When loading Large Memory Model programs, the operating system will automatically fixup all intersegment references, and adjust them according to the group base address. If the top bit of the byte (called the "Program Flag") at offset 007Fh in the 128-byte CMD file Header Record is set to 1, relocation information is present. In that case, the word at offset 007Dh of the CMD file Header Record holds the number of the first 128-byte record in the file holding relocation data. There is one relocation entry per segment reference (e.g., "CALLF", "SEG <segname>" in assembler). These relocation items are automatically included into the CMD file by LINK-86 when intersegment references are detected in object files.

Each entry consists of four bytes. The top four bits of the first byte specify the source group number, the bottom four bits specify the destination group number. An entry of 00h signals the end of the relocation list. The group numbers are identical to the numbers in the 9-byte Group Descriptors at the beginning of the CMD file Header Record. An entry of 12h would be generated, for example, if a "MOV AX, SEG variable" is found in the source of the program, and "variable" is in a Data Segment in the DGROUP of the program.

The second and third byte are a word in LSB-first format, which needs to be added to the segment address of the source segment. The fourth byte specifies the offset within this paragraph that needs to be relocated. If the code group is based at segment 1000h in our example and the 2nd, 3rd, and 4th bytes are 27h, 09h, and 0Dh, then the word at offset 1927h:000Dh will be relocated.

The relocation occurs by adding the segment address of the destination group to the word specified by the relocation item. If the data segment in our example was located at 3000h, then the value 3000h would be added to the word 1927h:0000Dh.

This relocation scheme is slightly more flexible than the MS-DOS scheme which assumes that all groups are contiguous in memory, therefore only permitting the last group to be of variable size.

HEADER.WS4 by Emmanuel ROCHE

Everything you wanted to know about the Header Record, but were too afraid to ask...

The more I work with CP/M-86 Plus ComManD files, the more often I need to know what is inside their Header Records.

For the record, a CMD file generally has 2 or 3 parts, which can be described thus:

```
+---------------+
| Header Record |
+---------------+
| Segments Used |
+---------------+
| Footer Record |  (If RSX(s) present(s))
```

```
      +---------------+
```

The Header Record contains a description of the Segments used by the program, that the loader of the CP/M-86 Plus Operating System will use to allocate memory and load the program in the TPA. As its name implies, it is 128 bytes long, or one record. It contains Group Descriptors at the beginning. A first byte of zero in a Group Descriptor indicates that no more Group Descriptors follows. The rest of the record is filled with null bytes (00h), except the 5 last bytes, which will be explained later.

The Segments Used part of the CMD file are those segments, up to 8.

The Footer Record is a record containing the names or the addresses/names of the RSXs linked/attached (up to 8) to this CMD file. This record does not exist if the CMD file has no RSX.

Enough theory, let us see some Header Records.

```
A>mbasic header
BASIC-86 Rev. 5.22
[CP/M-86 Plus]
Copyright 1977-1982 (C) by Microsoft
Created:  5-Mar-82
62390 Bytes free

HEADER> Enter CMD File Name: ? CMD1

       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code) | 01h | 0018 | 0000 | 0018 | 0000 |
      +-------+--------+--------+--------+--------+

Offset of Footer Record: 0000
Offset of Fixups Record: 0000
Program Flag: 00

(8080 Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? CMD2

       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code) | 01h | 0008 | 0000 | 0008 | 0000 |
      +-------+--------+--------+--------+--------+
(Data) | 02h | 0018 | 0000 | 0018 | 0000 |
      +-------+--------+--------+--------+--------+

Offset of Footer Record: 0000
Offset of Fixups Record: 0000
Program Flag: 00
```

(Small Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? CMD3

```
       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code) |  01h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Data) |  02h  |  0018  |  0000  |  0018  |  0000  |
      +-------+--------+--------+--------+--------+
(Extra)|  03h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
```

Offset of Footer Record: 0000
Offset of Fixups Record: 0000
Program Flag: 00

(Compact Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? CMD4

```
       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code) |  01h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Data) |  02h  |  0018  |  0000  |  0018  |  0000  |
      +-------+--------+--------+--------+--------+
(Extra)|  03h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Stack)|  04h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
```

Offset of Footer Record: 0000
Offset of Fixups Record: 0000
Program Flag: 00

(Compact Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? CMDX1

```
       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code) |  01h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Data) |  02h  |  0018  |  0000  |  0018  |  0000  |
```

```
      +-------+--------+--------+--------+--------+
(Extra) |  03h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Stack) |  04h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.1) |  05h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
```

Offset of Footer Record: 0000
Offset of Fixups Record: 0000
Program Flag: 00

(Compact Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? CMDX2

```
       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code)  |  01h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Data)  |  02h  |  0018  |  0000  |  0018  |  0000  |
      +-------+--------+--------+--------+--------+
(Extra) |  03h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Stack) |  04h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.1) |  05h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.2) |  06h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
```

Offset of Footer Record: 0000
Offset of Fixups Record: 0000
Program Flag: 00

(Compact Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? CMDX3

```
       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code)  |  01h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Data)  |  02h  |  0018  |  0000  |  0018  |  0000  |
      +-------+--------+--------+--------+--------+
(Extra) |  03h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Stack) |  04h  |  0008  |  0000  |  0008  |  0000  |
```

```
      +-------+--------+--------+--------+--------+
(Aux.1) |  05h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.2) |  06h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.3) |  07h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
```

Offset of Footer Record: 0000
Offset of Fixups Record: 0000
Program Flag: 00

(Compact Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? CMDX4

```
       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code)  |  01h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Data)  |  02h  |  0018  |  0000  |  0018  |  0000  |
      +-------+--------+--------+--------+--------+
(Extra) |  03h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Stack) |  04h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.1) |  05h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.2) |  06h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.3) |  07h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
(Aux.4) |  08h  |  0008  |  0000  |  0008  |  0000  |
      +-------+--------+--------+--------+--------+
```

Offset of Footer Record: 0000
Offset of Fixups Record: 0000
Program Flag: 00

(Compact Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? CALLVERS

```
       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code)  |  01h  |  0002  |  0000  |  0002  |  0000  |
      +-------+--------+--------+--------+--------+
(Data)  |  02h  |  0012  |  0000  |  0012  |  0000  |
```

```
      +-------+--------+--------+--------+--------+

Offset of Footer Record: 0004
Offset of Fixups Record: 0000
Program Flag: 00

(Small Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? ECHOVERS

       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code) |  01h  |  0006  |  0000  |  0006  |  0000  |
      +-------+--------+--------+--------+--------+
(Data) |  02h  |  0012  |  0000  |  0012  |  0000  |
      +-------+--------+--------+--------+--------+

Offset of Footer Record: 0400
Offset of Fixups Record: 0000
Program Flag: 10

(Small Memory Model)

Ok
run

HEADER> Enter CMD File Name: ? TESTGIOS

       G-Form  G-Length  A-Base   G-Min    G-Max
      +-------+--------+--------+--------+--------+
(Code) |  01h  |  0298  |  0000  |  0298  |  0000  |
      +-------+--------+--------+--------+--------+
(Data) |  02h  |  0058  |  0000  |  0058  |  0000  |
      +-------+--------+--------+--------+--------+
(Extra) |  03h  |  0006  |  0000  |  0006  |  0000  |
      +-------+--------+--------+--------+--------+
(Stack) |  04h  |  0094  |  0000  |  0094  |  0000  |
      +-------+--------+--------+--------+--------+

Offset of Footer Record: 0000
Offset of Fixups Record: 0073
Program Flag: 80

(Large Memory Model)

Ok
```

From the above, some things become clear: when there is only one Code Segment, it is a "8080 Memory model CMD file".

When there are 2 segments (but only if they are Code and Data), then it is a

"Small Memory model CMD file".

When there are more than 2 segments, it is a "Compact Memory model CMD file".

Historically, those were the 3 kinds of Memory Models described in the CP/M-86 Version 1.x technical documentation by Digital Research.

Under CP/M-86 Plus, RSXes exist. ECHOVERS is an example of such an RSX. As you can see, its Program Flag contains 10h. CALLVERS, the CMD file calling it, has its "Offset of Footer Record" field containing the value 0400h.

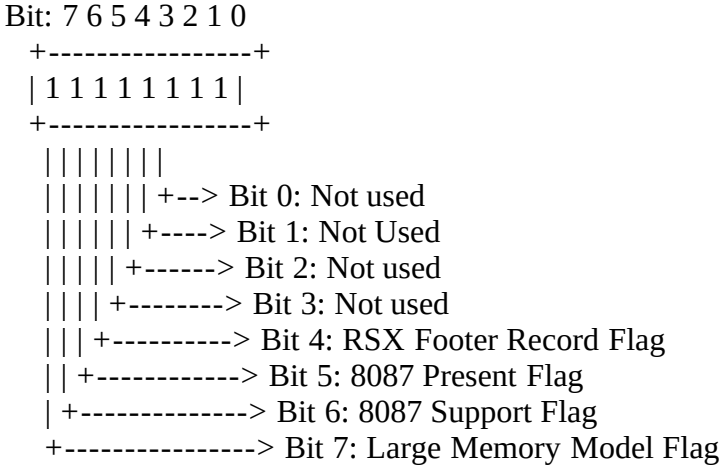
But, what is this "Large Memory model CMD file", TESTGIOS, we have seen last?

With Concurrent CP/M Version 1.0, Digital Research introduced in the Summer of 1983 a new kind of CMD file, where each segment can be up to 1 Megabyte in length... (Previously, they could only be 64k long, at the maximum.)

But, to be compatible with all the CMD files produced until then, they had to choose a way to let the CMD file loader of the operating system knows that this CMD file was of the new kind.

Since they could not modify the format of the Group Descriptors, they chose to use a byte inside the Header Record.

Over the years, this byte came to be used as flags for 4 things:



Program Flag format

The last 2 CMD files that we examined (ECHOVERS and TESTGIOS) displayed 10h and 80h; that is to say: Bits 4 and 7 set. But what is the purpose of Bits 5 and 6?

Searching for clues in later Digital Research guides, I finally found the following paragraph in the "Concurrent CP/M 3.1 Programmer's Guide":

3.1.2 8087 support

Concurrent CP/M provides optional 8087 support for systems that use the 8087 processor. This support is indicated by the Program Flag, byte 127 (7Fh), of

the CMD file Header Record. Setting bit 6 (bit 0 is least significant bit) of the Program Flag indicates optional 8087 support, which means that, if the 8087 is present, the program uses it; otherwise, the program will emulate it. If bit 5 of the Program Flag is set, it indicates that the 8087 must be present in order for the program to run. If no 8087 is present and bit 5 of the Program Flag is set, the system returns an error when it tries to load the program. The CHSET utility can be used to set the program's Header Record for optional or required 8087 support.

So, you now know what would mean a Program Flag with a value of 20h or 40h.

Now that we have explained everything known about the Header Record, here is the BASIC program used to learn all this:

```
list
10 REM HEADER.BAS  by Emmanuel ROCHE
20 :
30 PRINT
40 INPUT "HEADER> Enter CMD File Name: " ; file$
50 PRINT
60 file$ = file$ + ".CMD"
70 :
80 group$ (1) = "(Code) "
90 group$ (2) = "(Data) "
100 group$ (3) = "(Extra)"
110 group$ (4) = "(Stack)"
120 group$ (5) = "(Aux.1)"
130 group$ (6) = "(Aux.2)"
140 group$ (7) = "(Aux.3)"
150 group$ (8) = "(Aux.4)"
160 :
170 ruler$ =  "+-------+--------+--------+--------+--------+"
180 :
190 PRINT TAB(9) " G-Form  G-Length  A-Base   G-Min    G-Max"
200 PRINT TAB(9) ruler$
210 :
220 OPEN "R", #1, file$
230 FIELD #1, 9 AS GD$ (1), 9 AS GD$ (2), 9 AS GD$ (3), 9 AS GD$ (4),
   9 AS GD$ (5), 9 AS GD$ (6), 9 AS GD$ (7), 9 AS GD$ (8),
   51 AS zeroe$, 2 AS fo$, 2 AS fi$, 1 AS fl$
240 GET #1
250 :
260 i = i + 1
270 gform$ = LEFT$ (GD$ (i), 1)
280 gform = ASC (gform$)
290 IF gform = 0 THEN GOTO 440
300 NrGD = NrGD + 1
310 PRINT group$ (gform) TAB(9) "|  " ;
320 PRINT RIGHT$ ("0" + HEX$ (gform), 2) "h  |  " ;
330 glength$ = MID$ (GD$ (i), 2, 2)
340 PRINT RIGHT$ ("000" + HEX$ (CVI (glength$) ), 4) "  |  " ;
350 abase$ = MID$ (GD$ (i), 4, 2)
360 PRINT RIGHT$ ("000" + HEX$ (CVI (abase$) ), 4) "  |  " ;
```

```
370 gmin$ = MID$ (GD$ (i), 6, 2)
380 PRINT RIGHT$ ("000" + HEX$ (CVI (gmin$) ), 4) "  |  " ;
390 gmax$ = MID$ (GD$ (i), 8, 2)
400 PRINT RIGHT$ ("000" + HEX$ (CVI (gmax$) ), 4) "  |"
410 PRINT TAB(9) ruler$
420 GOTO 260
430 :
440 PRINT
450 PRINT "Offset of Footer Record: " ;
460 PRINT RIGHT$ ("000" + HEX$ (CVI (fo$) ), 4)
470 PRINT "Offset of Fixups Record: " ;
480 PRINT RIGHT$ ("000" + HEX$ (CVI (fi$) ), 4)
490 PRINT "Program Flag: " ;
500 fl = ASC (fl$)
510 PRINT RIGHT$ ("0" + HEX$ (fl), 2)
520 :
530 PRINT
540 PRINT "(" ;
550 IF fl > &H7F THEN PRINT "Large" ; : GOTO 590
560 IF NrGD = 1 THEN PRINT "8080" ; : GOTO 590
570 IF NrGD = 2 THEN PRINT "Small" ; : GOTO 590
580 IF NrGD > 2 THEN PRINT "Compact" ;
590 PRINT " Memory Model)"
600 :
610 PRINT
620 CLOSE
630 END
```

Initial program environment

When the operating system loads a program, there are certain system variables that are set to default or initial values before control is transferred to the program. The system also initializes certain information in a region referred to as the Base Page, which can be used by the program while executing. This section discusses the default or initial values of the system variables and Base Page variables at program load.

System default values

The following system variables have been discussed in the previous paragraphs, and are listed in Table 5-3 to indicate their default values at program load time.

Table 5-3. System variables and default values

Format: System variable
Load time value

Time and Date
The Time and Date can be set by the operator by using the DATE utility (see

the "CP/M-86 Plus User's Guide") or by the program by using the T_SET (Function 104) or S_SYSVAR (Function 49) system calls. The current Time and Date setting is the default value.

Default Password
This value can be set by the operator by using the SET utility (see the "CP/M-86 Plus User's Guide") or by the program by using the F_PASSWD (Function 106) system call. If set by the operator or the program, this password remains the default password until modified again.

Console Page Mode
The Console Page Mode is initially set by the system to display a page at a time. The Console Page Mode can be set by the operator by using the SETDEF command (see the "CP/M-86 Plus User's Guide"), or by the program by using the S_SYSVAR (Function 49) system call. The current Console Page Mode is the default mode.

Console Mode
At program load, the Console Mode is set to its default value, zero.

Output Delimiter
At program load, the Output Delimiter is set to its default value, the dollar sign ("$").

DMA Buffer
At program load, the DMA Buffer base is set to the address of the program's Data Segment, and the DMA Buffer offset is set to 0080h. This provides access to the default buffer in the Base Page.

Current Disk
At program load, this is the disk indicated at the command prompt, and is referred to as the "default disk". The application program can set this to another disk.

Current User Number
At program load, this is the user number indicated with the command prompt (equals 0 if not displayed). The application program can set the user number to another number, but when the program terminates, the CCP sets the Current User Number back to the CCP Current User Number.

Multisector Count
At program load, this value is always set to 1.

File System Error Mode
At program load, this value is always set to "Display and Terminate" error mode.

Program Return Code
At program load, this value is set to zero by the operating system.

Base Page initialization

The Base Page is the region of memory located from offset 0000h through 00FFh, relative to the DS register. When the system receives a request to load a program, it sets the following default values in the Base Page for access by the transient program. Table 5-4 describes the contents and locations of the data in the Base Page region. Locations are offsets from the DS register.

Table 5-4. Base Page structure

Format: Byte offset, Field name
Description

0000-0002h, CL
Code region length stored in low, middle, and high-order positions (following Intel storage convention). This value represents the length of the code group.

0003-0004h, CB
Base paragraph segment of the code group.

0005h, M80
M80 byte: a flag set to 1 if the 8080 Memory Model was used during load. Otherwise, the M80 byte is set to 0.

0006-0008h, DL
Data area length stored in low, middle, and high-order positions (following Intel storage convention). This value represents the length of the data group.

0009-000Ah, DB
Base paragraph segment of the data group.

000Bh
Reserved.

000C-000Eh, EL
Extra region length stored in low, middle, and high-order positions (following Intel storage convention). This value represents the length of the extra group.

000F-0010h, EB
Base paragraph segment of the extra group.

0011h
Reserved.

0012-0014h, SL
Stack region length stored in low, middle, and high-order positions (following Intel storage convention). This value represents the length of the stack group.

0015-0016h, SB
Base paragraph segment of the stack group.

0017h
Reserved.

0018-0029h, AUX
Region set aside for the four optional independent groups that might be required for programs that execute using the Compact Memory Model. These groups are stored as the previous groups, with 3 bytes for the group length, followed by 2 bytes for the base paragraph segment of that group, and 1 reserved byte. The initial values for these Group Descriptors are derived from the Header Record of the memory image file (CMD file) generated by the LINK-86 utility.

0030-004Fh
Reserved.

0050h, DR
Identifies the drive from which the transient program was loaded. Zero designates the default drive, while a value of 1 to 16 corresponds to drives A through P, respectively.

0051-0052h, PWO1
Contains the offset relative to the DS register of the password field of the first command-tail operand in the default DMA buffer beginning at 0080h. The system sets this field to zero if no password for the first command-tail operand is specified.

0053h, PWL1
Contains the length of the password field for the first command-tail operand. The system also sets this field to zero if no password for the first command-tail is specified.

0054-0055h, PWO2
Contains the offset relative to the DS register of the password field of the second command-tail operand in the default DMA buffer beginning at 0080h. The system sets this field to zero if no password for the second command-tail operand is specified.

0056h, PWL2
Contains the length of the password field for the second command-tail operand. The system also sets this field to zero if no password for the second command-tail is specified.

0057-005Bh
Reserved.

005C-007Bh, FCB1
Default File Control Block, FCB, area 1 initialized by the system from the first command-tail operand of the command line, if it exists.

006C-007Bh, FCB2
Default File Control Block, FCB, area 2 initialized by the system from the second command-tail operand of the command line, if it exists.

Note: FCB area 2 overlays the last 16 bytes of the 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.

007C, CR
Current record position of default FCB area 1. This field is used with default FCB area 1 in sequential record processing.

007D-007Fh, RRN1
Optional default random record position. This field is an extension of default FCB area 1 used in random record processing.

0080-00FFh, DMA Buffer
Default 128-byte DMA buffer. This buffer is also filled with the command tail when the system loads a transient program.

The system initializes the Base Page prior to initiating a transient program. The fields at 0050h and above are initialized from the command line invoking the transient program. The "CP/M-86 Plus User's Guide" describes the command line format in detail.

If a drive is specified in the command field, the system initializes byte 0050h to the drive number, A=1, B=2, ..., P=16. Otherwise, it sets byte 0050h to zero.

If a command tail is entered, the field FCB1 is initialized. Otherwise, bytes 005Ch and 0068h through 006Bh are set to zeroes, and the bytes from 005Dh through 0067h are set to blanks. If a password is specified for the first file specification in the command tail, the system stores the password information in the PWO1 and PWL1 fields. Otherwise, the system sets these fields to zeroes.

If a second file specification exists in the command tail, the system stores the file information in field FCB2. Otherwise, the bytes at 006Ch and 0078h-007Bh are set to zeroes, and bytes from 005Dh-0067h are set to blanks. If a password is specified for the second file specification of the command tail, the system places the password information in fields PWO2 and PWL2. Otherwise, these fields are set to zero.

Transient programs often use the default FCB at FCB1 for file operations. This FCB can even be used for random file access, because the three bytes RRN1 are available for this purpose. However, a transient program must first copy the contents of the default FCB at FCB2 to another area before using the default FCB at FCB1, because an open operation for the default FCB at FCB1 overwrites the FCB data at FCB2.

The default DMA buffer offset for transient programs is 0080h. At program load, the system initializes this area to contain the command tail of the command line. The first position contains the number of characters in the command tail, followed by a leading blank and the command tail characters. The system stores a binary zero (00h) after the last command tail character. The command tail characters are translated to ASCII uppercase. If the program uses the default DMA buffer for file I/O, it must extract the command tail information before performing any file operations.

Fields PWO1, PWL1, PWO2, and PWL2 contain password information for 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 buffer segment, before performing file operations.

Here is the output of a program showing the Base Page fields.

```
A>mbasic basepage
BASIC-86 Rev. 5.22
[CP/M-86 Plus]
Copyright 1977-1982 (C) by Microsoft
Created:  5-Mar-82
62390 Bytes free

Base Page areas
---------------

1F5F:0000| EF 70 00  Last position of Code group
1F5F:0003| 50 18     Base   of Code group
1F5F:0005| 00        M80 byte

1F5F:0006| EF FF 00  Last position of Data group
1F5F:0009| 5F 1F     Base   of Data group
1F5F:000B| 00        Unused 1

1F5F:000C| 00 00 00  Length of Extra group
1F5F:000F| 00 00     Base   of Extra group
1F5F:0011| 00        Unused 2

1F5F:0012| 00 00 00  Length of Stack group
1F5F:0015| 00 00     Base   of Stack group
1F5F:0017| 00        Unused 3

1F5F:0018| 00 00 00  Length of Auxiliary group #1
1F5F:001B| 00 00     Base   of Auxiliary group #1
1F5F:001D| 00        Unused 4

1F5F:001E| 00 00 00  Length of Auxiliary group #2
1F5F:0021| 00 00     Base   of Auxiliary group #2
1F5F:0023| 00        Unused 5

1F5F:0024| 00 00 00  Length of Auxiliary group #3
1F5F:0027| 00 00     Base   of Auxiliary group #3
1F5F:0029| 00        Unused 6

1F5F:002A| 00 00 00  Length of Auxiliary group #4
1F5F:002D| 00 00     Base   of Auxiliary group #4
1F5F:002F| 00        Unused 7

Dump of Reserved Area 1:

1F5F:0030| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1F5F:0040| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................

Drive from which the program was loaded:
```

```
1F5F:0050| 00       (Default drive)

1F5F:0051| 00 00    Offset of first password
1F5F:0053| 00       Length of first password

1F5F:0054| 00 00    Offset of second password
1F5F:0056| 00       Length of second password

Dump of Reserved Area 2:

1F5F:0057| 00 00 00 20 FF |... .

Dump of FCB1:

1F5F:005C| 00 42 41 53 45 50 41 47 45 20 20 20 00 00 00 00 |.BASEPAGE   ....

Dump of FCB2:

1F5F:006C| 00 20 20 20 20 20 20 20 20 20 20 20 00 00 00 00 |.           ....

1F5F:007C| 20       Current record position for FCB1

1F5F:007D| D00F16    Optional random record position for FCB1

Dump of DMA Buffer:

1F5F:0080| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1F5F:0090| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................
1F5F:00A0| 00 00 00 00 00 00 00 00 00 A2 20 91 20 22 4C 65 |.......... . "Le
1F5F:00B0| 6E 67 74 68 22 20 3B 00 14 0E 7C 01 91 20 22 20 |ngth" ;...|.. "
1F5F:00C0| 6F 66 20 22 20 47 52 4F 55 50 4E 41 4D 45 24 20 |of " GROUPNAME$
1F5F:00D0| 28 49 29 00 43 0E 86 01 91 20 53 45 47 24 20 FF |(I).C.... SEG$ .
1F5F:00E0| 82 20 28 22 30 30 30 22 20 F2 20 FF 9A 20 28 41 |. ("000" . .. (A
1F5F:00F0| 44 52 20 F2 20 14 29 2C 20 15 29 20 22 7C 20 22 |DR . .), .) "| "
```

Here is the source code of the program:

```
10 REM BASEPAGE.BAS  by Emmanuel ROCHE  for CP/M-86 Plus
20 :
30 PRINT
40 PRINT "Base Page areas"
50 PRINT "---------------"
60 PRINT
62 :
65 nb = 21
70 :
80 DEF SEG
90 HN$ = RIGHT$ ("0" + HEX$ (PEEK (10)), 2)
100 LN$ = RIGHT$ ("0" + HEX$ (PEEK (9)), 2)
110 SEG$ = HN$ + LN$ + ":"
120 :
130 DATA 8
```

```
140 ' Group name, M80/unused
150 DATA "Code group", "M80 byte"
160 DATA "Data group", "Unused 1"
170 DATA "Extra group", "Unused 2"
180 DATA "Stack group", "Unused 3"
190 DATA "Auxiliary group #1", "Unused 4"
200 DATA "Auxiliary group #2", "Unused 5"
210 DATA "Auxiliary group #3", "Unused 6"
220 DATA "Auxiliary group #4", "Unused 7"
230 :
240 READ NE  ' Number of Entries
250 FOR I = 1 TO NE
260     READ GROUPNAME$ (I), UNUSED$ (I)
270     GOSUB 310
280 NEXT I
290 GOTO 500
300 :
310 IF I = 6 THEN WHILE INKEY$ = "" : WEND
320 PRINT SEG$ RIGHT$ ("000" + HEX$ (ADR), 4) "| " ;
330 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 0)), 2) " " ;
340 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 1)), 2) " " ;
350 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 2)), 2) " " ;
360 PRINT TAB(nb) ;
370 IF I < 3 THEN PRINT "Last position" ; : ELSE PRINT "Length" ;
380 PRINT " of " GROUPNAME$ (I)
390 PRINT SEG$ RIGHT$ ("000" + HEX$ (ADR + 3), 4) "| " ;
400 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 3)), 2) " " ;
410 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 4)), 2) " " ;
420 PRINT TAB(nb) "Base   of " GROUPNAME$ (I)
430 PRINT SEG$ RIGHT$ ("000" + HEX$ (ADR + 5), 4) "| " ;
440 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 5)), 2) " " ;
450 PRINT TAB(nb) UNUSED$ (I)
460 PRINT
470 ADR = ADR + 6
480 RETURN
490 :
500 PRINT "Dump of Reserved Area 1:"
510 PRINT
520 SA = &H30 : EA = &H4F : GOSUB 560
530 PRINT
540 GOTO 740
550 :
560 IF SA > EA THEN PRINT CHR$ (7) ; "Dump error: start > end." : END
570 ALPHA$ = ""
580 PRINT SEG$ RIGHT$ ("000" + HEX$ (SA), 4) "| " ;
590 FOR I = 1 TO 16
600     AL = PEEK (SA)
610     AL$ = CHR$ (AL)
620     PRINT RIGHT$ ("0" + HEX$ (AL), 2) " " ;
630     IF (AL < &H20) OR (AL > &H7E) THEN AL$ = "."
640     ALPHA$ = ALPHA$ + AL$
650     SA = SA + 1
660     IF SA > EA THEN PRINT "|" ALPHA$ : GOTO 720
670 NEXT I
```

```
680 PRINT "|" ALPHA$
690 IF INKEY$ = CHR$ (32) THEN WHILE INKEY$ = "" : WEND
700 IF INKEY$ = CHR$  (3) THEN END
710 IF SA <= EA THEN GOTO 570
720 RETURN
730 :
740 PRINT "Drive from which the program was loaded:"
750 PRINT
760 ADR = &H50
770 PRINT SEG$ RIGHT$ ("000" + HEX$ (ADR), 4) "| " ;
780 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR)), 2) TAB(22) ;
790 IF PEEK (ADR) = 0 THEN PRINT "(Default drive)" ELSE PRINT "(Drive " CHR$ (PEEK (ADR) + &H40) ")"
800 PRINT
810 :
820 WHILE INKEY$ = "" : WEND
830 :
840 DATA 2
850 DATA first, second
860 :
870 READ NP  ' Number of Passwords
880 ADR = &H51
890 FOR I = 1 TO NP
900     READ PASSWORD$ (I)
910     GOSUB 950
920 NEXT I
930 GOTO 1060
940 :
950 PRINT SEG$ RIGHT$ ("000" + HEX$ (ADR), 4) "| " ;
960 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 0)), 2) " " ;
970 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 1)), 2) " " ;
980 PRINT TAB(nb) "Offset of " PASSWORD$ (I) " password"
990 PRINT SEG$ RIGHT$ ("000" + HEX$ (ADR + 2), 4) "| " ;
1000 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 2)), 2) " " ;
1010 PRINT TAB(nb) "Length of " PASSWORD$ (I) " password"
1020 PRINT
1030 ADR = ADR + 3
1040 RETURN
1050 :
1060 PRINT "Dump of Reserved Area 2:"
1070 PRINT
1080 SA = &H57 : EA = &H5B : GOSUB 560
1090 PRINT
1100 :
1110 PRINT "Dump of FCB1:"
1120 PRINT
1130 SA = &H5C : EA = &H6B : GOSUB 560
1140 PRINT
1150 :
1160 PRINT "Dump of FCB2:"
1170 PRINT
1180 SA = &H6C : EA = &H7B : GOSUB 560
1190 PRINT
1200 :
1210 ADR = &H7C
```

```
1220 PRINT SEG$ RIGHT$ ("000" + HEX$ (ADR), 4) "| " ;
1230 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR)), 2) " " ;
1240 PRINT TAB(nb) "Current record position for FCB1"
1250 PRINT
1260 :
1270 ADR = &H7D
1280 PRINT SEG$ RIGHT$ ("000" + HEX$ (ADR), 4) "| " ;
1290 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 0)), 2) ;
1300 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 1)), 2) ;
1310 PRINT RIGHT$ ("0" + HEX$ (PEEK (ADR + 2)), 2) ;
1320 PRINT TAB(nb) "Optional random record position for FCB1"
1330 PRINT
1340 :
1350 WHILE INKEY$ = "" : WEND
1360 :
1370 PRINT "Dump of DMA Buffer:"
1380 PRINT
1390 SA = &H80 : EA = &HFF : GOSUB 560
1400 PRINT
1410 :
1420 END
```

Transient program load and exit

Section 1, "CP/M-86 Plus system overview", discussed the procedure for loading and executing a transient program. After the operating system finds the file to be loaded, it initializes the DMA buffer with the command line, then calls P_CHAIN to initialize the Base Page values, and load and execute the program.

After control is transferred to the transient program, the program can use the 96-byte default stack and optionally return directly to the system upon program termination by executing a RETF (Far Return) instruction. The program can also terminate by using the P_TERMCPM (Function 0) or P_TERM (Function 143) system call. The program can also be terminated (or aborted) by the console operator by typing a Ctrl-C at the system console.

Chain to program

The system allows the transient program to specify the next command to be executed by making a P_CHAIN (Function 47) system call. The system executes the command specified by the transient program, instead of prompting the console for the next command.

Note that the P_CHAIN function also uses the Drive Search Chain when searching for the program to load.

Program Return Code

Transient programs can set the Program Return Code before terminating by making a P_CODE (Function 108) system call. The operating system initializes the Program Return Code to zero. The transient program that terminates successfully can use the Program Return Code to pass a value to a chained program. If the program terminates as a result of a fatal system error, or a Ctrl-C entered at the console aborted the program, the system sets the Program Return Code to an unsuccessful value. All other types of program termination leave the Program Return Code at its current value.

Explicit program load

The system allows multiple programs to reside in memory. A transient program such as a debugger can load additional programs for execution under its own control. The P_LOAD (Function 59) system call is used to load a program or a special resident system extension (RSX) module.

Resident System Extensions (RSXs)

Resident System Extensions (RSXs) are special CMD or RSX files that can be used by the programmer to modify or extend the functionality of the system. RSX modules intercept system calls, and either process them or pass them through to the BDOS for normal processing. More than one RSX module can be in memory at a time. A transient program can also use the P_RSX (Function 60) system call to call an RSX for special functions. P_RSX is a general purpose function that allows customized interfaces between programs and RSXs.

The order in which the RSX modules are loaded affects the order in which they intercept system calls. A more-recently loaded RSX receives control before a previously loaded RSX. Thus, if two RSXs are in memory at the same time, the more-recently loaded RSX handles the intercepted function first.

The RSX program code must include a prefix with the format as shown in Table 5-5.

Table 5-5. RSX prefix format

Offsets	Contents
0000-0002h	Jump to start of RSX program.
0003h	Terminate flag If non-zero, the RSX is removed from memory when the associated program terminates, or when P_CHAIN is called. Note: RSXs belonging to background programs are always removed when the program terminates.
0004-0007h	Next RSX in chain Offset is followed by segment address.
0008-000Fh	8-character RSX name Must follow standard file naming conventions.
0010-0011h	Data Segment of this RSX
0012h	Program ID of this RSX

0013-0014h (Not documented)
0015-001Fh Reserved
0020-????h RSX code

This consists of function number trapping and RSX function processing. If the function code in register CL is not intended for the RSX module, the RSX must execute a JMPF (Far Jump) instruction to the next RSX in the chain. Otherwise, it processes the function, and returns to the calling program by executing a RETF (Far Return) instruction.

Section 7, "Resident System Extension (RSXs)", includes an example of an RSX program and the steps required to generate the CMD file using the RSX program. It also describes the GENRSX utility, which generates the CMD and RSX files required to load and/or execute RSX modules.

Memory management

The CP/M-86 Plus system supports four types of memory allocation/de-allocation functions: load memory for transient programs, dynamic requests from transient programs, permanent (sticky) memory requests from transient programs, and memory required for the command loader.

Memory descriptors in a system memory descriptor table represent free or allocated areas of contiguous memory. This table has room for 32 memory descriptor entries. This table is initialized at system generation, and optionally at BIOS initialization.

The system allocates memory on a first fit basis. If there is not enough available memory or not enough available entries in the memory descriptor table, the system returns an error code.

A transient program can use the following system calls to allocate and free memory. Section 6, "System calls", describes these calls in detail.

Table 5-6. Memory allocation system calls

Mnemonic	Function	Description
MC_ABSALLOC	56	Allocate exact amount of memory at absolute location
MC_ABSMAX	54	Allocate no more than specified amount of memory at an absolute location.
MC_ALLOC	55	Allocate exact amount of memory
MC_MAX	53	Allocate no more than the specified amount of memory
MC_FREE	57	Free one previous allocation, or all dynamic allocations.

Load memory

When the system loads a transient program, it allocates memory according to the Group Descriptors in its CMD file Header Record. Usually, the system needs

only one memory descriptor to load a transient program. The memory allocation is less than or equal to the sum of the G_MAX fields, and greater than or equal to the sum of the G_MIN fields for all of the non-absolute groups defined in the CMD file Header Record. However, if the Header Record contains groups to be loaded at absolute locations, the system uses a memory descriptor for each absolute request to record the allocation. The use of absolute Group Descriptors is not encouraged, and is supported for compatibility and special uses, such as PROM programming.

The system also allocates memory for appended RSXs. Each appended RSX requires a memory descriptor.

When the program terminates or executes a P_CHAIN (Function 47) system call, the system releases the program's load memory.

Dynamic and sticky memory management

When a program requires many small dynamically allocated regions, the program should allocate a large region, and perform its own management of this area. Run-time libraries of most high-level programming languages manage memory in this manner.

For compatibility, the MC_FREE (Function 57) system call allows memory to be released at the end of a previously allocated area. The system cannot release memory at the beginning of, or in the middle of, an allocated area. Releasing entire memory allocations at once is encouraged over partial de-allocation, since future CP/M operating systems might not support partial de-allocation of memory regions.

A sticky memory allocation is made by setting a value in the Memory Control Block (MCB), as described in Section 6, "System calls". The program can release sticky memory only by using the MC_FREE (Function 57) system call. Once a program allocates sticky memory, it "owns" the sticky memory, and is the only program that can free this memory. However, after the program terminates, any program can de-allocate sticky memory if it knows the start of the sticky memory area.

A Resident System Extension provides an example of the use of sticky memory. The CP/M-86 Plus loader allocates sticky memory for RSXs, and puts the segment value of the RSX data in the RSX prefix. When a program terminates or calls the P_CHAIN (Function 47) system call, the system releases the RSX memory if the RSX is flagged for termination.

Sticky memory also allows for the loading of interrupt-driven routines that need to stay in memory.

Background tasking

CP/M-86 Plus allows up to four programs to run simultaneously: one program in the foreground, and three in the background. The background capability is

intended to run programs such as those that handle printer spooling, communications, and monitoring of instruments. Background programs must have the CMD file type, and should follow the rules described below.

In the foreground

When a program is initially loaded, it runs in the foreground. Console I/O is only permitted in the foreground, so any tasks the program has to perform that may require a dialogue with the user should be done before entering the background. While it is possible to do command line syntax checking, file I/O, memory allocation, etc, in the background, these tasks are better performed while it is still possible to give the user information about error conditions that may arise.

How to enter the background

Having completed any necessary dialogue with the user, the program need only make the CP/M-86 Plus C_DETACH (Function 147) system call in order to switch from foreground to background execution. For example:

```
Enter_Background:
;----------------
;
; Entry: None
; Exit : AX = 0000h if success, 0FFFFh if failure
;
      MOV     CL, 147        ; = C_DETACH
      INT     224            ; = BDOS
      RET                    ; If AX zero, now in background
```

For programmers writing their code in programming languages such as C, Pascal, COBOL, etc, this subroutine could be assembled separately, and then linked in with the main program.

The C_DETACH (Function 147) system call will fail if the BIOS does not support multitasking (no tick). See the S_SYSVAR (Function 49) system call.

The C_DETACH (Function 147) system call forces a new program environment to be created, this is always the Console Command Processor (CCP.CMD) running in the foreground. The RSXs loaded with the program become part of the background program environment. Previously loaded RSXs remain part of the foreground environment. For example, program P is loaded with an RSX that intercepts the BDOS calls. If P subsequently enters the background, the RSX will become part of the background environment, and will only intercept BDOS calls made by P, not BDOS calls made by any foreground program.

RSXs attached to background programs are always removed from memory when the background program terminates.

In the background

Console I/O is no longer possible. If a background program attempts to read from the system console, it will be terminated. Attempts to write to the console are ignored.

If necessary, the program can communicate with the user by directing its output to a console output file (filetype BAC, normally produced by the BACK.CMD utility), which the user can examine when the program has terminated. Note that, since there is no file locking under CP/M-86 Plus, it is the user's responsibility to ensure that two or more programs do not try to write to the same file at the same time. (Any number of programs may safely read from the same file at the same time.)

Only BDOS system calls are available to background programs.

General notes

If a program has to do character I/O, whether to the system console or to the auxiliary port, it should wherever possible do so via those BDOS functions that do not return until the character has been transferred. A program that sits in a loop polling the port status will consume processor time that may be better spent elsewhere. A user will notice that his system becomes sluggish when he loads his background program that polls the auxiliary status.

By default, the processor divides its time between the foreground and background programs in the ratio 16:1. For every cycle it spends processing a background program, it spends 16 cycles processing a foreground program. The user can alter this ratio with the built-in command FORE, which is documented in the "CP/M-86 Plus User's Guide".

EOF

CPMPPG6.WS4 (= "CP/M-86 Plus Programmer's Guide", section 6)

(Edited by Emmanuel ROCHE.)

Section 6: System calls

This section describes the interface conventions which allow the transient program to access system functions. It also discusses each system call in detail. This section also describes and lists the return and error codes for the system calls.

The system calls are categorized into functional groups. Table 6-1 defines these groups.

Table 6-1. The main system call groupings

Prefix	Category
A_	Auxiliary device I/O system calls
C_	Console I/O system calls
DRV_	Drive-related system calls
F_	File-access system calls
L_	List device I/O system calls
MC_	Memory management system calls
P_	Program execution system calls
S_	System-related system calls
T_	Time system calls

Table 6-2 lists the system calls alphabetically.

Table 6-2. CP/M-86 Plus system calls

Mnemonic	Function	Definition
Auxiliary device I/O system calls		
A_READ	3	Auxiliary input
A_STATIN	7	Auxiliary input status
A_STATOUT	8	Auxiliary output status
A_WRITE	4	Auxiliary output
A_READBLK	172	Auxiliary block input
A_WRITEBLK	173	Auxiliary block output
Console I/O system calls		
C_DELIMIT	110	Get/Set Current Output Delimiter
C_DETACH	147	Detach program from console
C_MODE	109	Get/Set Console Mode
C_RAWIO	6	Direct console I/O
C_READ	1	Console input

C_READSTR 10 Read console buffer
C_STAT 11 Get console status
C_WRITE 2 Console output
C_WRITEBLK 111 Write block of characters to console
C_WRITESTR 9 Write string of characters to console

Drive-related system calls

DRV_ALLOCVEC 27 Get drive allocation vector
DRV_ALLRESET 13 Reset all drives
DRV_DPB 31 Get disk parameter block address
DRV_FLUSH 48 Flush data buffers
DRV_FREE 39 Free drive
DRV_GET 25 Get default drive
DRV_GETLABEL 101 Get directory label data byte
DRV_LOGINVEC 24 Return drives logged in vector
DRV_RESET 37 Reset drive
DRV_ROVEC 29 Return drives Read-Only vector
DRV_SET 14 Set (select) drive
DRV_SETLABEL 100 Set directory label
DRV_SETRO 28 Set drive to Read-Only
DRV_SPACE 46 Get free space on drive

File-access system calls

F_ATTRIB 30 Set file's attributes
F_CLOSE 16 Close file
F_DELETE 19 Delete file
F_DMAGET 52 Get DMA segment and offset address
F_DMAOFF 26 Set DMA offset address
F_DMASEG 51 Set DMA segment address
F_ERRMODE 45 Set file system error mode
F_MAKE 22 Make a new file
F_MULTISEC 44 Set Multisector Count for file Read/Write
F_OPEN 15 Open file
F_PARSE 152 Parse filename
F_PASSWD 106 Set default password
F_RANDREC 36 Return record number for file Read/Write
F_READ 20 Read sequential record from file
F_READRAND 33 Read random record from file
F_RENAME 23 Rename file
F_SETDATE 116 Set File Stamps
F_SFIRST 17 Search for first file entry
F_SIZE 35 Compute file size
F_SNEXT 18 Search for next file entry
F_TIMEDATE 102 Return file time/date stamps and password mode
F_TRUNCATE 99 Truncate rest of file
F_USERNUM 32 Set/Get directory user number
F_WRITE 21 Write sequential record into file
F_WRITERAND 34 Write random record into file
F_WRITEXFCB 103 Write file's XFCB
F_WRITEZF 40 Write random record with zero fill

List device I/O system calls

L_WRITE	5	Write a character to LST:
L_WRITEBLK	112	Write a block of characters to LST:
Memory management system calls		
MC_ABSALLOC	56	Allocate absolute exact memory
MC_ABSMAX	54	Allocate absolute maximum memory
MC_ALLFREE	58	Free all memory
MC_ALLOC	55	Allocate exact amount of memory
MC_FREE	57	Free memory
MC_MAX	53	Allocate maximum memory
Program execution system calls		
P_CHAIN	47	Chain to next program
P_CODE	108	Set program return code
P_DELAY	141	Delay program
P_DISPATCH	142	Relinquish processor
P_LOAD	59	Load CMD or RSX module
P_RSX	60	Call Resident System Extension
P_TERM	143	Terminate calling program
P_TERMCPM	0	Terminate calling program
Miscellaneous system calls		
S_BDOSVER	12	Return BDOS Version Number
S_BIOS	50	Direct BIOS calls
S_SERIAL	107	Return Serial Number
S_SYSDAT	154	Get System Data address
S_SYSVAR	49	Get/Set System Variables
Time system calls		
T_GET	105	Get date and time
T_SET	104	Set date and time

System call parameters

When a program makes a system call, it loads values into the registers shown in Table 6-3, and initiates an Interrupt 224 (via the "INT 224" instruction) reserved by the Intel Corporation for CP/M.

Each system call is determined by a function code (number) which defines the function request. These function codes have been assigned function names (mnemonics) which you may find easier to remember.

The calling program passes the function code in register CL, with byte parameters in DL, and word parameters in DX. The system returns single-byte values in AL, word values in AX, and double-word values in ES and AX. On all returns, register BX is set to the value of register AX. The system saves all segment registers upon entry, and restores them upon exit, except when ES is

used for the segment part of a double-word returned value.

Table 6-3. Registers used in system calls

Reg.	Entry parameter
CL	Function code
DL	Byte parameter, or
DX	Word parameter, or
DX	Address -- Offset, and
DS	Address -- Segment

Reg.	Returned values
AL	Byte return, or
AX	Word return, or
AX	Address -- Offset, and
ES	Address -- Segment
BX	Same as AX
CX	Error code
AH	Return Code

The system also returns error codes in register CX. Table 6-4 lists CX error codes with the corresponding functions. All CP/M-86 Plus system calls return CX = 0000h on successful execution.

Table 6-4. CX error codes and corresponding functions

Dec	Hex	Function(s)	Error report
0	0000	--	Successful return
2	0002	--	Illegal function
3	0003	P_LOAD, MC_calls	Memory for allocation not available
4	0004	DEV_WAITFLAG	Illegal system flag number
5	0005	--	Flag overrun
6	0006	DEV_WAITFLAG	Flag underrun
12	000C	C_DETACH	No free process environments
18	0012	MC_calls	No available Memory Descriptors
23	0017	P_CHAIN,P_LOAD,F_PARSE	Illegal drive number
24	0018	P_CHAIN,P_LOAD,F_PARSE	Illegal filename
25	0019	P_CHAIN,P_LOAD,F_PARSE	Illegal filetype
29	001D	P_LOAD	Error reading file
30	001E	P_CHAIN, P_LOAD	Could not open file
32	0020	C_DETACH, MC_FREE	Not owner of resource
33	0021	P_LOAD	No code group descriptor in CMD file Header Record.
38	0026	P_CHAIN,P_LOAD,F_PARSE	Illegal password
41	0029	P_LOAD	Error in performing load-time fixups
42	002A	P_CHAIN, P_LOAD	Error loading RSX module
43	002B	MC_calls	Illegal parameter
45	002D	C_DETACH	No tick interrupt
46	002E	MC_LOAD	8087 in use, cannot load another program that uses the 8087.

Auxiliary device I/O system calls

The auxiliary device I/O system calls handle I/O operations for the logical devices AUXIN: and AUXOUT:. Table 6-5 lists all the auxiliary device I/O system calls discussed in this section.

Table 6-5. Auxiliary device I/O system calls

Mnemonic	Function	Description
A_READ	3	Auxiliary input
A_STATIN	7	Auxiliary input status
A_STATOUT	8	Auxiliary output status
A_WRITE	4	Auxiliary output
A_READBLK	172	Auxiliary block input
A_WRITEBLK	173	Auxiliary block output

BDOS Function 3: A_READ (Auxiliary Input)

Entry Parameters:
Register CL: 3

Returned Values:
Register AL: ASCII character
BL: Same as AL

Function
The A_READ system call reads the next 8-bit character from the logical auxiliary input device AUXIN: into register AL. Control does not return to the calling program until the character is read.

BDOS Function 7: A_STATIN (Auxiliary Input Status)

Entry Parameters:
Register CL: 7

Returned Values:
Register AL: Auxiliary Input Status (0FFh = Ready, 00h = Not ready)
BL: Same as AL

Function
The A_STATIN system call checks the input status of the logical auxiliary input device AUXIN:.

System action
The system returns the value 0FFh in register AL if a character is ready for input from the logical device AUXIN:. If no character is ready for input, the system returns the value 00h in register AL.

BDOS Function 8: A_STATOUT (Auxiliary Output Status)

Entry Parameters:
Register CL: 8

Returned Values:
Register AL: Auxiliary Output Status (0FFh = Ready, 00h = Not ready)
BL: Same as AL

Function
The A_STATOUT system call checks the output status of the logical auxiliary output device AUXOUT:.

System action
The system returns the value 0FFh in register AL if the logical auxiliary device AUXOUT: is ready for output. If the device is not ready for output, the system returns the value 00h in register AL.

BDOS Function 4: A_WRITE (Auxiliary Output)

Entry Parameters:
Register CL: 4
AL: ASCII character

Function
The A_WRITE system call sends the ASCII character in register AL to the logical auxiliary output device AUXOUT:. Control does not return to the calling program until the device is ready for output.

BDOS Function 172: A_READBLK (Auxiliary Block Input)

Entry Parameters:
Register CL: 172
DX: CHCB Address -- Offset

Returned Values:
Register AX: Number of characters read
BX: Same as AX

Function
The A_READBLK system call reads characters from the logical auxiliary input device AUXIN: and writes them into the character buffer located by the CHaracter Control Block (CHCB) addressed by DX.

The format of the CHCB is as follows:

Bytes 0 and 1: Offset of character buffer
Bytes 2 and 3: Segment of character buffer
Bytes 4 and 5: Length of character buffer (word value)

This system call returns the number of characters actually read from the default auxiliary device in register AX. A_READBLK returns to the calling

process when the status of AUXIN: indicates that the device is empty, or the character buffer is full. This call does not return control to the calling process until at least one character has been read.

BDOS Function 173: A_WRITEBLK (Auxiliary Block Output)

Entry Parameters:
Register CL: 173
DX: CHCB Address -- Offset

Returned Values:
Register AX: Number of characters written
BX: Same as AX

Function
The A_WRITEBLK system call sends the character string located by the CHaracter Control Block (CHCB) addressed in register DX to the logical auxiliary device AUXOUT:.

The format of the CHCB is as follows:

Bytes 0 and 1: Offset of character string
Bytes 2 and 3: Segment of character string
Bytes 4 and 5: Length of character string (word value)

A_WRITEBLK returns the number of characters written to the default auxiliary device in register AX. This system call returns to the calling process when the status of AUXOUT: indicates that the device is full, or the character string has been written. A_WRITEBLK does not return control to the calling process until at least one character has been written.

Console I/O system calls

The console I/O system calls handle I/O operations for the logical console CON:. Table 6-6 lists all the console I/O system calls discussed in this section.

Table 6-6. Console I/O system calls

Mnemonic	Function	Description
C_DELIMIT	110	Get/Set Current Output Delimiter
C_DETACH	147	Detach console from program
C_MODE	109	Get/Set Console Mode
C_RAWIO	6	Direct console I/O
C_READ	1	Console input
C_READSTR	10	Read console buffer
C_STAT	11	Get console status
C_WRITE	2	Console output
C_WRITEBLK	111	Write block of characters to console
C_WRITESTR	9	Write string of characters to console

BDOS Function 110: C_DELIMIT (Get/Set Output Delimiter)

Entry Parameters:
Register CL: 110
DX: 0FFFFh (Get), or
DL: Output Delimiter (Set)

Returned Values:
Register AL: Output Delimiter, or no value
BL: Same as AL

Function
The C_DELIMIT system call sets or returns the current Output Delimiter.

Entry condition
If register DX = 0FFFFh, the request is to return the Output Delimiter in register AL. Otherwise, the request is to set the current Output Delimiter.

System action
The system returns the Output Delimiter in register AL if register DX = 0FFFFh. Otherwise, the system sets the Output Delimiter to the value in register DL.

The default Output Delimiter is the dollar sign ("$"). The Output Delimiter is used by the C_WRITESTR (Function 9) system call.

BDOS Function 147: C_DETACH (Detach Program from Console)

Entry Parameters:
Register CL: 147

Returned Values:
Register AX: 0000h if successful, 0FFFFh if failure
BX: Same as AX
CX: Error Code (See Table 6-4)

Function
The C_DETACH system call detaches the program from the console.

System action
The system detaches the program from the console. This function is used if the program wishes to run in the background.

The system returns an error code if it cannot run any background programs.

BDOS Function 109: C_MODE (Get/Set Console Mode)

Entry Parameters:
Register CL: 109
DX: 0FFFFh (Get), or Console Mode (Set)

Returned Values:
Register AX: Console Mode, or no value
BX: Same as AX

Function
The C_MODE system call sets or returns the Console Mode (see Section 2, "Character device I/O").

Entry condition
If register DX = 0FFFFh, the request is to return the Console Mode in register AX. Otherwise, the request is to set the Console Mode to the value in register DX.

System action
The system returns the Console Mode in register AX if register DX = 0FFFFh. Otherwise, the system sets the Console Mode to the value in register DX.

The Console Mode is a 16-bit system variable described in Section 2, "Character device I/O", and summarized in Table 6-7.

Table 6-7. Console Mode variable bit summary

Bit	Function
0	1 indicates Ctrl-C only status for C_STAT. 0 indicates normal status function for C_STAT.
1	1 disables stop scroll; Ctrl-S, start scroll; Ctrl-Q. 0 enables stop scroll, start scroll support.
2	1 disables tab expansion and printer echo. 0 enables normal console output mode.
3	1 disables Ctrl-C system interception. 0 enables Ctrl-C system interception.

BDOS Function 6: C_RAWIO (Direct Console I/O)

Entry Parameters:
Register CL: 6
DL: 0FFh (input/status) or
0FEh (status) or
0FDh (input) or
char (output)

Returned Values:
(input/status)
Register AL: 00h (no character) or
Character
(status)
Register AL: 00h (no character) or
0FFh (character ready)

(input)
Register AL: Character
BL: Same as AL

Function
The C_RAWIO system call allows the calling program to process unedited ("direct" or "unadorned") console input or output.

System action
The calling program selects the type of Direct Console I/O by passing different values in register DL. Table 6-8 summarizes the C_RAWIO options and responses:

Table 6-8. C_RAWIO options and responses

Reg.DL	Meaning
0FFh	Get console input, or check console status. If a character has been typed, the system returns the character in register AL. Otherwise, the system sets register AL to 00h.
0FEh	Get console status. If a character has been typed, the system sets register AL to 0FFh. Otherwise, the system sets register AL to 00h.
0FDh	Get console input. If a character has been typed, the system returns it in register AL. Otherwise, the system waits until a character is typed, and returns it in register AL.
ASC	Output ASCII character to the console.

Note: The system disables all normal character control functions when C_RAWIO is used. See Section 2, "Character device I/O", for a description of control character functions.

BDOS Function 1: C_READ (Console Input)

Entry Parameters:
Register CL: 1

Returned Values:
Register AL: Character
BL: Same as AL

Function
The C_READ system call reads the next character from the console CONIN:.

System action
The system reads the next character typed at the keyboard, and returns it in register AL. If it is a graphic character or a Carriage Return, Line Feed, or BackSpace character, it is echoed to the console. All other characters are

read, but not echoed to the console. If printer echo (Ctrl-P) has been activated, the character is also sent to the list device LST:. The system expands tab characters (Ctrl-I) in columns of eight characters.

If the Console Mode is in default state (see Section 2, "Character device I/O"), the system intercepts Ctrl-S, Ctrl-Q, Ctrl-C, and Ctrl-P characters.

If start scroll/stop scroll is disabled, the system passes Ctrl-S and Ctrl-Q characters to the calling program. If Ctrl-P is disabled, the system passes Ctrl-P to the program.

If no character is typed at the keyboard, the system waits and does not return to the calling program until a character is typed.

BDOS Function 10: C_READSTR (Read Console Buffer)

Entry Parameters:
Register CL: 10
DX: BUFFER Address -- Offset: 0FFFFh
(If buffer to be used is current DMA address
and buffer is already initialized.)
DS: BUFFER Address -- Segment

Function
The C_READSTR system call reads a line of edited console input from console CONIN: to a buffer addressed by the DX register. If the DX register is 0FFFFh, the system displays the string in the current DMA buffer for operator editing. See the "User's Guide" for details of line-editing input.

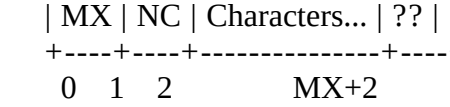


Figure 6-1. Console input buffer format

MX is the maximum number of characters the buffer holds (1 through 255), and is set by the program upon entry. If the MX field is set to zero, the system assumes the value one.

NC is the number of characters (0 through MX) placed in the buffer, and is set by C_READSTR.

CHARACTERS are the characters entered by the operator (DX <> 0FFFFh), or initialized by the calling program (DX = 0FFFFh).

BDOS Function 11: C_STAT (Get Console Status)

Entry Parameters:
Register CL: 11

Returned Values:

Register AL: 01h if character is ready, or
00h if character is not ready
BL: Same as AL

Function
The C_STAT system call checks to see if a character has been typed at the system console CONIN:.

System action
If a character has been typed at the console, the system returns the value in register AL. Otherwise, the system returns 00h in register AL.

Note: If bit 0 and bit 3 of the Console Mode are set (see Section 2, "Character device I/O", and the C_MODE (Function 109) system call), the system returns 01h only if Ctrl-C has been typed at the console. Otherwise, the system sets register AL to 00h.

BDOS Function 2: C_WRITE (Console Output)

Entry Parameters:
Register CL: 2
DL: 8-bit character

Function
The C_WRITE system call sends the 8-bit character in register DL to the console device CONOUT:.

System action
If the Console Mode is in the default state (see Section 2, "Character device I/O"), the system expands tab characters (Ctrl-I) in columns of 8 characters, checks for stop scroll, start scroll characters (Ctrl-S, Ctrl-Q), and sends the character to the list device LST: if printer echo (Ctrl-P) is activated. Otherwise, the system action is determined by the bits set in the Console Mode.

BDOS Function 111: C_WRITEBLK (Print Block)

Entry Parameters:
Register CL: 111
DX: CHCB Address -- Offset
DS: CHCB Address -- Segment

Function
The C_WRITEBLK system call sends the character block located by the Character Control Block (CHCB) to the console device CONOUT:.

System action
If the Console Mode is in the default state (see Section 2), the system expands tab characters (Ctrl-I) in columns of 8 characters, checks for stop scroll, start scroll characters (Ctrl-S, Ctrl-Q), and sends the character string to the list device LST: if printer echo (Ctrl-P) is activated. Otherwise, the system action is determined by the bits set in the Console

Mode.

The CHCB format is as follows:

Bytes 0-1: Offset of character string
Bytes 2-3: Segment of character string
Bytes 4-5: Length of character string (word value)

BDOS Function 9: C_WRITESTR (Print String)

Entry Parameters:
Register CL: 9
DX: String Address -- Offset
DS: String address -- Segment

Function
The C_WRITESTR system call sends the character string addressed by the DX register to the output console CONOUT:.

System action
If the Console Mode is in the default state (see Section 2, "Character device I/O"), the system expands tab characters (Ctrl-I) in columns of 8 characters, checks for stop scroll, start scroll characters (Ctrl-S, Ctrl-Q), and sends the character string to the list device LST: if printer echo (Ctrl-P) is activated.

The string is terminated by the Output Delimiter, which is normally the dollar sign ("$"), but a program can change the Output Delimiter to any other character by making a C_DELIMIT (Function 110) system call.

Assigning logical devices

Most CP/M-86 Plus computer systems have a console with a keyboard and screen display, and perhaps a printer. To keep track of these different input and output devices, CP/M-86 Plus assigns different physical devices to logical devices in the system.

A logical device represents a class of physical devices. For example, both the keyboard and auxiliary port are physical devices that can provide input. Console input, therefore, is a logical device that can be assigned to a physical device like the keyboard. Similarly, console output can be assigned to a physical device such as the screen. When a logical device is not assigned to anything specific, it is said to be assigned to a null, or dummy, device.

The following table gives the names and types of CP/M-86 Plus logical devices. It also shows the physical devices assigned to these logical devices in the standard CP/M-86 Plus system.

Logical Device Name	Logical Device Type	Physical Device Assignment

Logical Device Name	Logical Device Type	Physical Device Assignment
CONIN:	Console Input	Keyboard
CONOUT:	Console Output	Screen
AUXIN:	Auxiliary Input	Null
AUXOUT:	Auxiliary Output	Null
LST:	List Output	Printer

Typically, you can assign a number of physical devices to any given logical device.

The following table shows a number of possibilities.

Logical Device Name	Logical Device Type	Physical Device Assignment
CONIN:	Console Input	Keyboard
CONOUT:	Console Output	Screen, Printer
AUXIN:	Auxiliary Input	Modem, Lightpen, Joystick, Mouse, and so on
AUXOUT:	Auxiliary Output	Modem, Printer, Plotter, and so on
LST:	List Output	Printer, Screen

If you use your computer for a range of tasks, you might want to add different kinds of devices to your system. For example, a line printer, a modem, a lightpen, or even a joystick for playing games.

In some implementations of CP/M-86 Plus, you can change the standard assignments with a DEVICE command. If your system supports the DEVICE command, you can, for example, assign AUXIN: and AUXOUT: to a modem, so that your computer can communicate with other computers over the telephone.

Drive-related system calls

The drive-related system calls operate on the system logical drives. Table 6-3 lists all the drive-related system calls discussed in this section.

Table 6-13. Drive-related system calls

Mnemonic	Function	Description
DRV_ALLOCVEC	27	Get drive allocation vector
DRV_ALLRESET	13	Reset all drives
DRV_DPB	31	Get disk parameter block address
DRV_FLUSH	48	Flush data buffers
DRV_FREE	39	Free drive
DRV_GET	25	Get default drive
DRV_GETLABEL	101	Get directory label data byte
DRV_LOGINVEC	24	Return drives logged in vector
DRV_RESET	37	Reset drive
DRV_ROVEC	29	Return drives Read-Only vector

DRV_SET 14 Set (select) drive
DRV_SETLABEL 100 Set directory label
DRV_SETRO 28 Set drive to Read-Only
DRV_SPACE 46 Get free space on drive

BDOS Function 27: DRV_ALLOCVEC (Get Address of Allocation Vector)

Entry Parameters:
Register CL: 27

Returned Values:
Register AX: ALLOCVEC Address -- Offset
BX: Same as AX
ES: ALLOCVEC Address -- Segment

Function
The DRV_ALLOCVEC system call returns the base address of the allocation vector for the currently selected drive.

System action
The system returns in register ES and AX the base address of the allocation vector for the currently selected drive. Bits in the allocation vector denote allocated blocks when set to 1, and unallocated blocks when set to 0. The high-order bit in the first byte of the allocation vector corresponds to block 0. The DRV_SPACE system call can be used to get the number of free 128-byte records on a drive.

BDOS Function 13: DRV_ALLRESET (Reset Disk System)

Entry Parameters:
Register CL: 13

Function
The DRV_ALLRESET system call restores the file system to a reset state where all the disk drives are set to Read/Write, the default drive is set to drive A, and the default DMA address is reset to offset 0080h relative to the current DMA segment address.

BDOS Function 31: DRV_DPB (Get Address of DPB Parameter Block)

Entry Parameters:
Register CL: 31

Returned Values:
Register AX: DPB Address -- Offset (0FFFFh on Physical Errors)
BX: Same as AX
ES: DPB Address -- Segment

Function
The DRV_DPB system call returns the address of the BIOS-resident Disk Parameter Block for the currently selected drive. The calling program can use

this address to display or compute the space on the drive.

System action
The system returns in register AX the offset address of the Disk Parameter Block, and the segment address in register ES.

If the system encounters a physical error, the system sets register AX to 0FFFFh.

The Disk Parameter Block (DPB) contains the parameters that define the actual disk. See the "System Guide" for complete description of the DPB.

BDOS Function 48: DRV_FLUSH (Flush Buffers)

Entry Parameters:
Register CL: 48
DL: Purge Flag

Returned Values:
Register AL: Return Code (0FFh = Physical Error)
AH: Physical Error Code (00h = No error)
BX: Same as AX

Function
The DRV_FLUSH system call forces the write of any write-pending records contained in internal blocking/deblocking buffers.

System action
The system writes any write-pending records contained in the internal blocking/deblocking buffers. If register DL is set to 0FFh, the system also purges all active data buffers belonging to the calling process. Programs that provide write with read operation need to purge internal buffers, to ensure that verifying data comes from the disk, and not from internal data buffers. The PIP utility is an example of such a program.

If the flush operation is successful, register AL is set to 00h. If a physical error is detected by the system and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH to one of the following physical error codes:

01: Disk I/O error
02: Read-Only disk
04: Invalid drive error

BDOS Function 39: DRV_FREE (Free Drive)

Entry Parameters:
Register CL: 39

Returned Values:
Register AL: 00h

BL: Same as AL

Function
The DRV_FREE system call is a function that is not supported under CP/M-86 Plus. If called, the file system returns a 00h in register AL, to indicate that the free request is successful.

BDOS Function 25: DRV_GET (Return Current Disk)

Entry Parameters:
Register CL: 25

Returned Values:
Register AL: Drive Number (0-15)
BL: Same as AL

Function
The DRV_GET system call returns the currently-selected default disk or drive in register AL.

System action
The system returns in register AL the currently-selected default drive. The drive number ranges from 0 through 15, corresponding to drives A through P, respectively.

BDOS Function 101: DRV_GETLABEL (Return Directory Label Data)

Entry Parameters:
Register CL: 101
DL: Drive Number (0-15)

Returned Values:
Register AL: Directory Label Data Byte
(00h if no label, 0FFh if Physical Error)
AH: Physical Error Code
BX: Same as BX

Function
The DRV_GETLABEL system call returns the directory label data byte (see Section 3) of the directory label for the specified drive.

Entry parameter
The drive number in register DL ranges from 0 through 15, with 0 corresponding to drive A, 1 corresponding to drive B, and so on through 15 for drive P.

System response
If the disk contains a directory label, the system returns the directory label data byte in register AL. If the disk has no label, the system sets register AL to 00h. If the system encounters a physical error on the specified drive, and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, it sets register AL to 0FFh, and register AH to one of the following physical error codes:

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

BDOS Function 24: DRV_LOGINVEC (Return Login Vector)

Entry Parameters:
Register CL: 24

Returned Values:
Register AX: Login Vector
BX: Same as AX

Function
The DRV_LOGINVEC system call returns a bit map of currently logged in disk drives.

System action
The system returns in register AX the Login Vector. The Login Vector is a 16-bit value whose least significant bit corresponds to drive number 0, or drive A, and whose most significant bit corresponds to drive number 15, or drive P. A bit set to 0 indicates that the corresponding drive is not logged in, while a bit set to 1 indicates that the corresponding drive is logged in. A drive is logged in when its directory is read into the system, and its allocation vectors built. A drive can be logged in explicitly by using the DRV_SET (Function 14) system call, or implicitly by a file operation specifying a non-zero value in the DR field of the referenced FCB.

The Login Vector is a data structure of 16-bit values specifying one or more drives, where the least significant bit corresponds to drive A, and the most significant bit corresponds to the sixteenth drive, labeled P. Figure 6-2 illustrates the format of the Login Vector.

```
   +--------------------------------+
DRV | P O N M L K J I H G F E D C B A |
   +--------------------------------+
BIT   F E D C B A 9 8 7 6 5 4 3 2 1 0
```

Figure 6-2. Login Vector structure

BDOS Function 37: DRV_RESET (Reset Drive)

Entry Parameters:
Register CL: 37
DX: Drive Vector

Returned Values: None

Function
The DRV_RESET system call programmatically restores specified drives to the reset state. A reset drive is one that is not logged in, and is in Read/Write status.

System action
The system resets the drives specified by the drive vector in register DX. The drive vector is a 16-bit value whose least significant bit corresponds to drive 0 or A, and whose most significant bit corresponds to drive 15 or P. The system resets those drives whose corresponding bits are set to 1.

BDOS Function 29: DRV_ROVEC (Get Read-Only Vector)

Entry Parameters:
Register CL: 29

Returned Values:
Register AX: Read-Only Vector
BX: Same as AX

Function
The DRV_ROVEC system call returns a 16-bit value in register AX, which indicates which drives have the temporary Read-Only bit set.

System action
The system returns the Read-Only vector in register AX. This 16-bit value indicates which drives are set to Read-Only. The least significant bit corresponds to drive 0 or A, while the most significant bit corresponds to drive 15 or P.

A drive is set to Read-Only by using the DRV_SETRO (Function 28) system call.

The Read-Only Vector is a data structure of 16-bit values specifying one or more drives, where the least significant bit corresponds to drive A, and the most significant bit corresponds to the sixteenth drive, labeled P. Figure 6-3 illustrates the format of the Read-Only Vector.

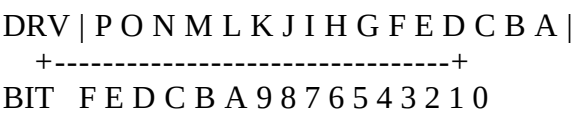


Figure 6-3. Read-Only Vector structure

BDOS Function 14: DRV_SET (Select Disk)

Entry Parameters:
Register CL: 14
DL: Drive Number (0-15)

Returned Values:
Register AL: Return Code
AH: 00h, or Physical Error Code
BX: Same as AX

Function

The DRV_SET system call designates the disk drive specified in register DL as the default disk for subsequent system file I/O operation.

System action
The system selects the specified disk drive in register DL as the default disk for subsequent file I/O operation. The value in register DL can range from 0 (for drive A) through 15 (for drive P). The system also logs in the selected drive if it is currently in the reset state. If the select operation is successful, the system sets register AL to 00h. If a physical error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and sets register AH to one of the following physical error codes:

01: Disk I/O error
04: Invalid drive

BDOS Function 100: DRV_SETLABEL (Set Directory Label)

Entry Parameters:
Register CL: 100
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Directory Code (0FFh if Physical Error)
AH: Physical or Extended Error Code (00h if no error)
BX: Same as AX

Function
The DRV_SETLABEL system call creates or updates a directory label for the specified drive.

Entry conditions
The calling program sets the F1 through F8 (name) and T1 through T3 (type) fields of the FCB to be used by the system as the directory label. The EX (extent field) of the FCB is also set, to define the user's specification of the directory label data byte, as described in Section 3.

If the current directory label is password-protected, the correct password must be placed in the first 8 bytes of the current DMA.

If bit 0 of the data byte is set, then the system is to assign a new password to the directory label. This new password is placed in the second 8 bytes of the current DMA.

System action
The system creates or updates the directory label with the information specified in the FCB. If successful, the system returns a Directory Code in register AL with the value 0 to 3, and sets register AH to 00h. If no space existed in the referenced directory to create a directory label, the system sets register AL to 0FFh, and register AH to 00h. If a physical or extended error is encountered and the File System Error Mode is in default mode, the

system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH 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 28: DRV_SETRO (Write Protect Disk)

Entry Parameters:
Register CL: 28

Returned Values:
Register AL: Return Code
BL: Same as AL

Function
The DRV_SETRO system call sets the currently-selected drive to Read-Only.

System action
The system sets the currently-selected drive to Read-Only. No program can write to a drive that is in Read-Only state. The DRV_RESET (Function 37) and DRV_ALLRESET (Function 13) system calls restore a Read-Only drive to Read/Write status. The system automatically restores a Read-Only drive to Read/Write status if a media change is detected on the drive.

If the write-protect operation is successful, the system sets register AL to 00h. Otherwise, it sets register AL to 0FFh.

BDOS Function 46: DRV_SPACE (Get Disk Free Space)

Entry Parameters:
Register CL: 46
DL: Drive Number

Returned Values:
Register AL: Return Code
AH: 00h, or Physical Error Code
BX: Same as AX

Function
The DRV_SPACE system call computes the number of free sectors (128-byte records) on the specified drive.

System action
The system returns a binary number in the first 3 bytes of the current DMA buffer. The first byte corresponds to the low byte, the second byte is the middle byte, and the third byte is the high byte of the binary number.

If the DRV_SPACE function is successful, it sets register AL to 00h. If a

physical error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and sets register AH to one of the following physical error codes:

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

File access system calls

Most file-access system calls reference a File Control Block (FCB). This data structure is described in detail in Section 3, "CP/M-86 Plus file system". The file-access system calls include calls that operate on files within a directory, calls that operate on records within files, and other miscellaneous functions related to file I/O. Table 6-14 lists all the file-access system calls.

Table 6-14. File-access system calls

Mnemonic	Function	Description
F_ATTRIB	30	Set file's attributes
F_CLOSE	16	Close file
F_DELETE	19	Delete file
F_DMAGET	52	Get DMA segment and offset address
F_DMAOFF	26	Set DMA offset address
F_DMASEG	51	Set DMA segment address
F_ERRMODE	45	Set file system error mode
F_MAKE	22	Make a new file
F_MULTISEC	44	Set Multisector Count for file Read/Write
F_OPEN	15	Open file
F_PARSE	152	Parse filename
F_PASSWD	106	Set default password
F_RANDREC	36	Return record number for file Read/Write
F_READ	20	Read sequential record from file
F_READRAND	33	Read random record from file
F_RENAME	23	Rename file
F_SFIRST	17	Search for first file entry
F_SIZE	35	Compute file size
F_SNEXT	18	Search for next file entry
F_TIMEDATE	102	Return file time/date stamps and password mode
F_TRUNCATE	99	Truncate rest of file
F_USERNUM	32	Set/Get directory user number
F_WRITE	21	Write sequential record into file
F_WRITERAND	34	Write random record into file
F_WRITEXFCB	103	Write file's XFCB
F_WRITEZF	40	Write random record with zero fill

BDOS Function 30: F_ATTRIB (Set File Attributes)

Entry Parameters:

Register CL: 30
DX: FCB Address -- Offset
DS: FCB Address -- Segment
Interface Attribute:
F6' = 0 --> Do not set byte count
F6' = 1 --> Set last record byte count

Returned Values:
Register AL: Return Code
AH: Physical or Extended Error
BX: Same as AX

Function
The F_ATTRIB system call modifies a file's attributes, and optionally sets its last record byte count.

Entry conditions
The file attributes F1' through F4' and T1', T2', and T3' can be set to 0 or 1 by the calling program. Section 3, "CP/M-86 Plus file system", describes these attributes.

The interface attribute F6' specifies if the byte count of the last record is to be set by F_ATTRIB. If the calling program sets F6' to 1 and the CS field of the FCB to the byte count, F_ATTRIB sets the last record byte count for the file. If F6' is 0, then F_ATTRIB does not record a last record byte count for the file.

In addition, if the specified file is password-protected, the correct password must be placed in the first 8 bytes of the current DMA buffer, or have been previously established as the Default Password (See the F_PASSWORD (Function 106) system call).

System action
The system searches the directory for entries belonging to the current user number that matches the file specified in the referenced FCB. The system then updates the file directory FCB, to set the attributes specified in the referenced FCB.

If F6' is set to 1, the last record byte count is also recorded in the file directory FCB.

If the F_ATTRIB function is successful, the system sets register AL and AH to 00h.

If the specified file is not found, the system sets registers AL and AH to 00h.

If a physical or extended error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and sets register AH 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 the FCB filename or filetype field

BDOS Function 16: F_CLOSE (Close File)

Entry Parameters:
Register CL: 16
DX: FCB Address -- Offset
DS: FCB Address -- Segment
Interface Attribute:
F5' = 0 --> Permanent close
F5' = 1 --> Partial close

Returned Values:
Register AL: Directory Code (0FFh = File not found)
AH: Physical Error Code (00h = No error)
BX: Same as AX

Function
The F_CLOSE system call permanently records new or updated information in the referenced FCB to the disk directory.

Entry conditions
The referenced FCB must have previously been initialized by a successful F_OPEN or F_MAKE system call. The calling program sets the interface attribute F5' to 0 to indicate a permanent close operation; that is to say, the program has completed file operations on the referenced file. F5' set to 1 requests a partial close operation; that is to say: the system updates the directory, but keeps the file in the open state.

System action
If the referenced FCB contains new information because of write operations, the system records the new information in the disk directory. If the FCB does not contain new information, the system does not update the directory.

If the close operation is successful, the system sets registers AL and AH to 00h.

If the file referenced in the FCB is not in the disk directory, the system sets register AL to 0FFh, and register AH to 00h.

If a physical or extended error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH to one of the following physical error codes:

01: Disk I/O error
02: Read-Only disk
04: Invalid drive error

BDOS Function 19: F_DELETE (Delete File)

Entry Parameters:
    Register CL: 19
        DX: FCB Address -- Offset
        DS: FCB Address -- Segment
    Interface Attribute:
        F5' = 0 --> Standard delete
        F5' = 1 --> Delete only XFCBs
    Password in DMA buffer (if required)

Returned Values:
    Register AL: Directory Code (0FFh = File not found)
        AH: Physical Error Code (00h = No error)
        BX: Same as AX

Function
The F_DELETE system call removes file directory entries and/or XFCBs that match the files referenced in the FCB.

Entry conditions
The files to be deleted must be specified in the filename and filetype fields of the FCB. The filename and filetype can contain ambiguous references; that is to say: question marks ("?") in bytes F1 through T3. If the interface attribute F5' is set to 0, the calling program requests removal of all file directory entries belonging to the files specified in the FCB. If F5' is set to 1, then the system is to delete only the XFCBs of the specified files.

If any of the files specified are password-protected, the correct password must be placed in the first 8 bytes of the current DMA buffer, or must have been previously established as the Default Password (see F_PASSWD (Function 106) system call).

System action
For standard delete operations, the system removes all directory entries belonging to the specified files in the referenced FCB. All disk and directory space owned by the deleted files are returned to free space, and become available for allocation to other files.

Directory XFCBs belonging to the deleted files are also removed from the directory.

If F5' is 1, the system deletes only the directory XFCBs of the files in the referenced FCB.

If the delete operation is successful, the system sets registers AL and AH to 00h.

If the file referenced in the FCB is not in the disk directory, the system sets register AL to 0FFh, and register AH to 00h.

For both delete operations, if any of the files fail the password check, or are Read-Only, or are currently open by another program, then the system does not delete any files or XFCBs.

If a physical or extended error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH 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 52: F_DMAGET (Return Address of DMA Buffer)

Entry Parameters:
Register CL: 52

Returned Values:
Register AX: DMA Address -- Offset
BX: Same as AX
ES: DMA Address -- Segment

Function
The F_DMAGET system call returns the current DMA buffer offset in register AX, and the base segment in register ES.

See Section 3, "CP/M-86 Plus file system", for a description of the DMA buffer.

BDOS Function 26: F_DMAOFF (Set DMA Buffer Offset)

Entry Parameters:
Register CL: 26
DX: DMA Address -- Offset

Function
The F_DMAOFF system call changes the current DMA buffer offset to another offset.

System action
The system sets the current DMA buffer offset to that specified in register DX.

See Section 3, "CP/M-86 Plus file system", for a description of the DMA buffer.

BDOS Function 51: F_DMASEG (Set DMA Buffer Segment)

Entry Parameters:
Register CL: 51
DX: DMA Address -- Segment

Function
The F_DMASEG system call sets the current DMA buffer segment.

System action
The system sets the current DMA buffer segment to that specified in register DX.

Note: Upon initial program load, the default DMA buffer segment base is set to the address of the program's data segment (the initial value of DS) and the DMA buffer offset to 0080h, which provides access to the default DMA buffer in the Base Page.

See Section 3, "CP/M-86 Plus file system", for a description of the DMA buffer.

BDOS Function 45: F_ERRMODE (Set System Error Mode)

Entry Parameters:
Register CL: 45
DL: System Error Mode
(0FFh = Return error mode)
(0FEh = Return and Display mode)
(Any other value = Default mode) (Display and Terminate)

Function
The F_ERRMODE system call determines the system action when physical and extended errors are encountered. See Section 3, "CP/M-86 Plus file system", for a description of the different Error Modes.

System action
The system sets the File System Error Mode to that specified in register DL. If register DL contains 0FFh, the system sets the Error Mode to the Return error mode. If DL contains 0FEh, the system sets the Error Mode to the Return and Display mode. Otherwise, the system sets the Error Mode to the Default mode (Display and Terminate).

BDOS Function 22: F_MAKE (Make File)

Entry Parameters:
Register CL: 22
DX: FCB Address -- Offset
DS: FCB Address -- Segment
Interface Attribute:
F6' = 0 --> No password (default)
F6' = 1 --> Assign password
Password in DMA buffer (if F6' = 1)

Returned Values:
Register AL: 00h (0FFh = Physical or Extended Error)
AH: Physical or Extended Error Code
(00h = No error, or no directory space)
BX: Same as AX

Function
The F_MAKE system call creates a new directory entry for the file specified in the FCB for the current user number, and activates the FCB for read and write operations.

Entry conditions
The calling program must have initialized the following fields in the FCB: the DR field for the drive, F1 through F8 and T1 through T3 fields specifying the filename and filetype, and the EX field set to 0 for the extent number. The CR field, the current record field, must be set to zero 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. If F6' is 0, then the system is not to assign a password. If F6' is 1, then 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 (Bit 7: Read mode, Bit 6: Write mode, Bit 5: Delete mode).

System action
The system creates a new directory entry for the specified file in the FCB. It also creates an XFCB for the file, if the referenced drive has enabled password protection, and the calling program has set F6' to 1.

The system initializes both the directory FCB and the referenced FCB to an empty file. The system also initializes all file attributes to zero. If date and time stamping is active on the specified drive, the system creates the date and time stamps for the file.

If the F_MAKE function is successful, the system sets registers AL and AH to 00h.

If there is no directory space available for another file entry, the system sets register AL to 0FFh, and register AH to 00h.

If the file referenced in the FCB is already in the disk directory, or if a physical error is detected and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH 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 the FCB filename or filetype field

BDOS Function 44: F_MULTISEC (Set Multisector Count)

Entry Parameters:
Register CL: 44
DX: 1-128 (Number of Sectors)

Returned Values:
Register AL: Return Code
BL: Same as AL

Function
The F_MULTISEC system call sets the Multisector Count for the system. The Multisector Count determines the number of 128-byte records that the system reads or writes with one call to any of the following system calls: F_READRAND, F_WRITERAND, F_READ, F_WRITE, and F_WRITEZF.

System action
The system sets the Multisector Count to the value in register DL.

Upon return, the system sets register AL to 00h if the value in register DL is in the range of 1 to 128. Otherwise, register AL is set to 0FFh.

Note: The default Multisector Count is 1. See Section 3, "CP/M-86 Plus file system", for a discussion of multisector I/O.

BDOS Function 15: F_OPEN (Open File)

Entry Parameters:
Register CL: 15
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Directory Code (0FFh = File not found, or Physical error)
AH: Physical Error Code (00h = No error)
BX: Same as AX

Function
The F_OPEN system call activates the FCB for the file specified in the referenced FCB for the current user number or user 0.

Entry conditions
The calling program sets the DR field, the F1 through F8 and T1 through T3 (filename and filetype) fields, and zeroes the EX extent field of the FCB.

If the file specified is password-protected in Read mode, the correct password is placed in the first 8 bytes of the current DMA buffer, or must have been previously established as the default password (see the F_PASSWORD (Function 106) system call).

The calling program sets the CR field of the FCB to 0FFh to request F_OPEN to return the last record byte count of the file. F_OPEN returns this value in the CR field.

System action
The system searches the directory for the file specified in the referenced FCB. If a match is found, the system copies the relevant directory information from the matching directory entry into the allocation field of the referenced (memory) FCB. The process of copying the directory information into the

referenced FCB is referred to as "activating" the FCB.

If the current user is non-zero and the file to be opened does not exist under the current user number, the system 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 and the system sets interface attribute F8' to 1, to indicate that the file can be accessed only in Read-Only mode.

If the file is password-protected in Write mode and the correct password was not passed in the DMA buffer or did not match the default password, the system sets interface attribute F7' to 1.

Write operations are not supported for an activated FCB if either interface attributes F7' or F8' are set to 1. If the open operation is successful, the system makes an access date and time stamp for the opened file if the following conditions are satisfied: 1) the referenced drive has a directory label that request access date and time stamping, and 2) the FCB EX field is zero.

If the open operation is successful, the system sets registers AL and AH to 00h.

If the file does not exist, the system sets register AL to 0FFh, and register AH to 00h.

If a physical or extended error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH 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 152: F_PARSE (Parse Filename)

Entry Parameters:
Register CL: 152
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AX: 0FFFFh = Error, 0000h = End of line or ASCII string
BX: Same as AX
CX: Error Code (See Table 6-4)

Function
The F_PARSE system call parses an ASCII file specification, and prepares a File Control Block (FCB) for the file.

Entry conditions
The calling program passes the address of a data structure, called the "Parse

Filename Control Block" (PFCB), in register DX. The PFCB contains the offset of the ASCII file specification, followed by the offset of the target FCB to be filled by the system, as shown below:

```
PFCB:   DW      Ifile          ; Offset of input ASCII filespec
        DW      TFCB           ; Offset of target FCB
```

The maximum length of the input ASCII filespec to be parsed is 128 bytes. The length of the target FCB must be 36 bytes. The system assumes the input ASCII filespec to be in the following form:

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

where those items enclosed in curly brackets ("{" and "}") are optional.

System action
The system parses the first file specification it finds in the input string. The system first eliminates leading blanks and tabs. The system assumes the file specification ends on the first delimiter it hits 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 whole file specification. The system call recognizes the following characters as delimiters:

space
tab
return
null
; (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)
[(left square bracket)
] (right square bracket)
/ (slant)

If the system reaches a non-graphic character (in the range 1 through 31) not listed above, it treats it as an error.

The system initializes the specified FCB as shown in Table 6-15.

Table 6-15. FCB initialization

Byte	Description
0	The drive field is set to the specified drive. If the drive is not specified, the default value is used. 0=default, 1=A, 2=B, and so on.
1-8	The name is set to the specified filename. Letters are converted to uppercase. If the name is not 8 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 with question marks ("?"). The system returns an error if the filename is more than 8 bytes long.

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 uppercase. 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 ("?"). The system returns an error if the type field is more than 3 bytes long.

12-15 Filled in with zeros.

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 not 8 characters long, remaining bytes are padded with blanks. All letters are converted to uppercase. The system returns an error if the password field is more than 8 bytes long. A blank in the first position of the password field implies no password is specified.

24-31 Reserved for system use.

If the system encounters an error, it sets all fields that have not been parsed to their default values, then returns 0FFFFh in register AX.

On a successful parse, the system checks the next item in the ASCII input string. It skips over trailing blanks and tabs, and looks at the next character. If the character is a null character (00h), it returns a 0000h in AX indicating the end of the ASCII input string. If the next character is a delimiter, it returns the address of the delimiter. If the next 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 ASCII input string is a null (00h) or a Carriage Return (0Dh), the system returns a 0000h in AX indicating the end of the string.

If the system is to be used to parse a subsequent filename in the ASCII input string, the returned address should be advanced over the delimiter before placing it in the PFCB.

BDOS Function 106: F_PASSWD (Set Default Password)

Entry Parameters:
Register CL: 106
DX: Password Address -- Offset
DS: Password Address -- Segment

Function
The F_PASSWD system call specifies a default password value to be used by the system for password-protected files.

System action
The system establishes the default password as the 8-byte field referenced by the Password Address.

When the system accesses a password-protected file, it checks the current DMA buffer and/or the default password for the correct password.

If either value matches the file's password, the system allows access to the file.

BDOS Function 36: F_RANDREC (Set Random Record Number)

Entry Parameters:
Register CL: 36
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
R0,R1,R2 field in FCB set

Function
The F_RANDREC system call sets the random record number of the next record to be accessed from a file that has been previously accessed sequentially.

System action
The system returns the random record number of the next record to be accessed in the R0, R1, and R2 field of the referenced FCB.

BDOS Function 20: F_READ (Read Sequential)

Entry Parameters:
Register CL: 20
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Return Code
AH: Physical Error, or Record Count
BX: Same as AX

Function
The F_READ system call reads the next 1 to 128 logical records (128-byte records) from a file into memory, beginning at the current DMA address. The system Multisector Count determines the number of records to be read (see Section 3, "CP/M-86 Plus file system"). The Multisector Count defaults to 1 record.

Entry conditions
Before reading a record from a file, the calling program must first activate the file FCB (using the F_OPEN (Function 15) or F_MAKE (Function 22) system call). This ensures that the FCB is properly initialized for subsequent read operations. If the intent is to read sequentially from the beginning of the

file, the calling program must set the CR (byte 32) current record field of the FCB to 00h.

System action
The system reads the record specified in the CR field of the FCB into the current DMA buffer address, and automatically increments the CR field to the next record position. If the CR field overflows, the system automatically opens the next logical extent (16 Kbytes), and resets the CR field to zero in preparation for the next read operation.

If the Multisector Count is greater than one, the system reads multiple consecutive records into memory, beginning at the current DMA buffer, and automatically increments the CR field of the FCB to read each record.

If the read operation is successful, the system sets register AL to 00h.

If an error condition is detected, the system sets register AL to one of the following error codes:

01: Reading unwritten data (end of file)
09: Invalid FCB
10: Media change occurred

and register AH to the number of records successfully read before the error occurred. This value can range from 0 to 127, depending on the value of the Multisector Count. This value is always 0 if the Multisector Count is 1.

If a physical error condition is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register Ah to one of the following physical error codes:

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

BDOS Function 33: F_READRAND (Read Random)

Entry Parameters:
Register CL: 33
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Return Code
AH: Physical Error, or Record Count
BX: Same as AX

Function
The F_READRAND system call reads the record whose record number is indicated by the value in the R0,R1,R2 field of the FCB.

Entry conditions
Before reading a random record from a file, the calling program must first

open the file (using F_OPEN (Function 15) system call) with the EX field in the FCB set to zero. This ensures that the FCB is properly initialized for subsequent random access operations. The calling program then sets the record number to be read in the random record field of the FCB. This field is indicated by the 24-bit value constructed from the three-byte (R0,R1,R2) field of the FCB. Note that the sequence of 24 bits is stored with the least significant byte first (R0), the middle byte next (R1), and the most significant 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, and 255 (or 0FFh) in both R0 and R1.

System action
The system reads the record into the current DMA buffer, and automatically sets the logical extent and current record values, but (unlike the sequential F_READ (Function 20) system call) the system does not increment the CR (current record) field. Thus, a subsequent F_READRAND system call re-reads the same record. After a random read operation, a file can be accessed sequentially, starting from the current record position. When switching from random to sequential mode, the last randomly-accessed record is re-read or re-written.

If the Multisector Count is greater than 1, the system reads multiple consecutive records into memory, beginning at the current DMA buffer address, and automatically increments the R0,R1,R2 field of the FCB to read each record. However, upon return to the calling program, the system restores the FCB's random record number to the original requested value.

If the read operation is successful, the system sets registers AL and AH to 00h.

If an error condition is detected, the system sets register AH to 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

and register AH to the number of records successfully read before the error occurred. This value can range from 0 to 127, depending on the value of the Multisector Count. This value is always 0 if the Multisector Count is 1.

If a physical error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH to one of the following error codes:

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

BDOS Function 23: F_RENAME (Rename File)

Entry Parameters:
Register CL: 23
DX: FCB Address -- Offset
DS: FCB Address -- Segment
Bytes 0-16 of FCB: Old filename
Bytes 17-27 of FCB: New filename
Bytes 0-7 of DMA: Password (if required)

Returned Values:
Register AL: 00h = No error, 0FFh = File not found, or Physical Error
AH: Physical or Extended Error Code (00h = No error)
BX: Same as AX

Function
The F_RENAME system call changes the filename and filetype of a disk's file.

Entry conditions
The calling program sets the DR drive field, filename, and filetype fields of the referenced FCB to specify the old filename, and bytes 17 to 27 (corresponding to bytes 1 to 11) to specify the new filename. If the file specified is password-protected, the correct password must be placed in the first 8 bytes of the current DMA buffer, or must have been previously established as the default password (see the F_PASSWD (Function 106) system call).

System action
The system changes all directory entries of the file specified by the first 16 bytes of the referenced FCB to the file specification in the second 16 bytes of the FCB. The system uses byte 0 to select the drive. The corresponding drive code at byte 16 is ignored.

If the rename operation is successful, the system sets registers AL and AH to 00h.

If the file does not exist, the system sets register AL to 0FFh, and register AH to 00h.

If a physical or extended error occurs and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH 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 the FCB filename or filetype field

BDOS Function 116: F_SETDATE (Set File Stamps)

Entry Parameters:

Register CL: 116
DX: FCB Address -- Offset
DS: FCB Address -- Segment
Time-Date stamp in DMA Buffer

Returned Values:
Register AL: Return Code
AH: Physical Error
BX: Same as AX

Function
The F_SETDATE system call sets the time and the date stamp fields for the specified file to the time and date stamp values specified in the first 8 bytes of the DMA buffer. The specified file must currently be open in Locked mode by the calling process.

(ROCHE> I *think* that those 8 bytes are the same as the first 8 bytes of an SFCB, that is to say:

Bytes 0 to 3: Create/Access stamp
Bytes 4 to 7: Update stamp

Where each stamp has the DAT format:

Byte 0 and 1: Date field
Byte 2: Hour field
Byte 3: Minute field
(There is no Seconds field.))

BDOS Function 17: F_SFIRST (Search for First)

Entry Parameters:
Register CL: 17
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Directory Code (0FFh = File not found, or Physical Error)
AH: Physical or Extended Error Code (00h = No error)
BX: Same as AX

Function
The F_SFIRST system call scans the directory for a match with the file specified in the FCB.

Entry conditions
The calling program must set the DR drive field, the F1 through F8 and T1 through T3 (filename and filetype) fields, and the EX extent field of the referenced FCB. The EX field must be set to 00h.

An ambiguous file reference can be specified in the FCB by a question mark ("?") in any of the bytes F1 through F8 and T1 through T3 in the FCB. A question mark matches all entries in the directory in the corresponding

position. This type of file reference is called an "ambiguous file reference", and can be used to search for multiple files in the directory.

If the DR field of the referenced FCB is set to a question mark ("?"), the system locates the first directory entry residing on the current default drive for any user number. If the DR field is set to a drive code, the system scans for the first matching file entry in the specified directory that belongs to the current user number.

System action
The system locates the first directory entry that matches the entry in the referenced FCB. It also initializes the F_SNEXT (Function 18) system call, in case the calling program wants to locate subsequent matching entries.

If the search is successful, that is to say: a match is found, the system returns a Directory Code in register AL with the value 0 to 3, and fills the buffer at the current DMA address with the record containing the directory entry, and its relative starting position is AL times 32. The directory information can be extracted from the buffer at this position.

If the search is not successful (the file is not found), the system sets register AL to 0FFh, and register AH to 00h.

If a physical error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH to one of the following physical error codes:

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

BDOS Function 35: F_SIZE (Compute File Size)

Entry Parameters:
Register CL: 35
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: 00h = File was found,
0FFh = File not found, or Physical Error
AH: Physical or Extended Error Code (00h = No error)
BX: Same as AX
R0,R1,R2 field of FCB: File size

Function
The F_SIZE system call determines the virtual file size of the file specified in the referenced FCB. This is the position of the record immediately following the end of the file. The virtual file size corresponds to the physical file 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 is written to a file using the F_WRITERAND (Function 33)

system call, the virtual size of the file is 262,144, even though only one data block is allocated to the file.

Entry conditions
The calling program sets the DR drive field, and the filename and filetype fields of the referenced FCB.

System action
The system sets R0,R1,R2 (the random record number) field of the FCB to the Random Record Number + 1 of the last record of the file. Note that, if the R2 byte is set to 04h, and R0 and R1 are both 00h, then the file contains the maximum record count: 262,144.

Note: A program can append data to the end of an existing file by using the F_SIZE system call to set the random record number position to the end of the file, then performing a sequence of random writes.

The file need not be open in order to use F_SIZE. However, if data has been written to the file, the file must be closed before calling F_SIZE. Otherwise, an incorrect file size might be returned by the system.

If the F_SIZE function is successful, the system sets registers AL and AH to 00h.

If the file does not exist, the system sets register AL to 0FFh, and register AH to 00h.

If a physical or extended error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH to one of the following physical or extended error codes:

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

BDOS Function 18: F_SNEXT (Search for Next)

Entry Parameters:
Register CL: 18

Returned Values:
Register AL: Directory Code (0FFh = File not found, or Physical Error)
AH: Physical or Extended Error Code (00h = No error)
BX: Same as AX

Function
The F_SNEXT system call continues the scan of the directory for the file specified in the FCB.

Entry conditions
The calling program must have previously executed an F_SFIRST (Function 17) or a previous F_SNEXT system call with no other intervening disk-related system

calls.

System action
The system locates the next directory entry of the file referenced in the specified FCB, and returns to the calling program, as described in the F_SFIRST system call explanation.

BDOS Function 102: F_TIMEDATE (Read File Date Stamps and Password Mode)

Entry Parameters:
Register CL: 102
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: 00h = No error, 0FFh = Physical Error
AH: Physical Error Code
BX: Same as AX

Function
The F_TIMEDATE system call returns the date and time stamp information and password mode for the specified file in byte 12 and bytes 24 through 32 of the referenced FCB.

System action
If the specified file is found, the system sets the following fields in the referenced FCB:

Byte 12: Password Mode field
Bit 7: Read mode
Bit 6: Write mode
Bit 5: Delete mode

Byte 12 equal to zero indicates the file has not been assigned a password.

Bytes 24 to 27: Create (or Access) time stamp field
Bytes 28 to 31: Update time stamp field

The time stamp fields are set to binary zeroes 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 the T_SET (Function 104) and T_GET (Function 105) system calls.

If the system finds the specified file in the directory, it sets registers AL and AH to 00h. If the specified file is not found, the system sets register AL to 0FFh, and register AH to 00h.

If a physical or extended error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH to one of the following physical or extended error codes:

01: Disk I/O error

04: Invalid drive error
09: "?" in the FCB filename or filetype field

BDOS Function 99: F_TRUNCATE (Truncate File)

Entry Parameters:
Register CL: 99
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: 00h = File was found
0FFh = File not found, or Physical Error
AH: Physical or Extended Error Code (00h = No error)
BX: Same as AX

Function
The F_TRUNCATE system call sets the last record of a file to the random record number specified in the referenced FCB.

Entry conditions
The calling program sets the DR drive field, the filename and filetype fields of the referenced FCB to specify the file to be truncated, and R0,R1,R2 (the random record number) of the FCB to the record number to which the file is to be truncated.

If the file specified is password-protected, the correct password must be placed in the first 8 bytes of the current DMA buffer, or have been previously established as the default password (see the F_PASSWD (Function 106) system call).

F_TRUNCATE requires that the file specified in the FCB be closed.

Also, the random record number field must 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 to which data has actually been written.

System action
The system sets the last record number of the specified file to the random record number in the referenced FCB.

If the F_TRUNCATE function is successful, the system sets registers AL and AH to 00h.

If the file does not exist, the system sets register AL to 0FFh, and register AH to 00h.

If a physical or extended error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH 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 the FCB filename or filetype field

BDOS Function 32: F_USERNUM (Set/Get User Number)

Entry Parameters:
Register CL: 32
DL: User Number (Set), or
0FFh (Get User Number)

Returned Values:
Register AL: Current User Number (if Get)
BL: Same as AL

Function
The F_USERNUM system call returns the Current User Number, or sets the current user number.

System action
The system returns the Current User Number in register AL if register DL = 0FFh. Otherwise, the system sets the Current User Number to the value specified in register DL (modulo 16). The Current User Number can range from 0 to 15.

BDOS Function 21: F_WRITE (Write Sequential)

Entry Parameters:
Register CL: 21
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Return Code
AH: Physical Error, or Record Count
BX: Same as AX

Function
The F_WRITE system call writes 1 to 128 128-byte records to a file from memory beginning at the current DMA buffer address. The system Multisector Count determines the number of records to be written (see Section 3, "CP/M-86 Plus file system"). The Multisector Count defaults to 1 record.

Entry conditions
Before writing a record to a file, the calling program must first activate the file FCB (using F_OPEN (Function 15) or F_MAKE (Function 22) system call) with the EX (extent) field in the FCB set to 00h. This ensures that the FCB is properly initialized for subsequent write operations. If the intent is to write sequentially from the beginning of the file, the calling program must

set the CR field of the FCB to 00h.

System action
The system writes the record specified in the CR field of the FCB from the current DMA buffer address, and automatically increments the CR field to the next record position. If the CR field overflows, then the system automatically opens the next logical extent, and resets the CR field to 00h, in preparation for the next write operation.

If the write operation is successful, the system sets registers AL and AH to 00h.

If an error condition is detected, the system sets register AL to one of the following error codes:

01: No available directory space
02: No available data block
09: Invalid FCB
10: Media change occurred

and register AH to the number of records successfully read before the error occurred. This value can range from 0 to 127, depending on the value of the Multisector Count. This value is always 0 if the Multisector Count is 1.

If a physical error condition is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH to one of the following physical or extended 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

BDOS Function 34: F_WRITERAND (Write Random)

Entry Parameters:
Register CL: 34
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Return Code
AH: Physical Error, or Record Count
BX: Same as AX

Function
The F_WRITERAND system call writes the record whose record number is indicated by the values in the R0,R1,R2 (random record) field of the FCB.

Entry conditions

Before writing a random record to a file, the calling program must first activate the file FCB (using F_OPEN (Function 15) or F_MAKE (Function 22) system call), with the EX extent field in the FCB set to 00h. This ensures that the FCB is properly initialized for subsequent random access operations. The calling program then sets the record number to be written in the random record field of the FCB. This field is indicated by the 24-bit value constructed from the three-byte (R0,R1,R2) field beginning at position 33 of the FCB. Note that the sequence of 24 bits is stored with the least significant byte first (R0), the middle byte next (R1), and the most significant 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, and 0FFh in bytes R0 and R1.

System action
The system writes the record from the current DMA buffer address, and automatically sets the logical extent and current record values, but (unlike the sequential F_WRITE (Function 22) function) the system does not increment the CR (current record) field of the FCB. Thus, a subsequent F_WRITERAND system call rewrites the same record.

After a random write operation, a file can be accessed sequentially, starting from the current record position. However, the last randomly-accessed record is rewritten when switching from random to sequential mode.

If the write operation is successful, the system sets registers AL and AH to 00h.

If an error condition is detected, the system sets register AL to 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

and register AH to the number of records successfully read before the error occurred. This value can range from 0 to 127, depending on the value of the Multisector Count. This value is always 0 if the Multisector Count is 1.

If a physical error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH to one of the following physical or extended 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

BDOS Function 103: F_WRITEXFCB (Write File XFCB)

Entry Parameters:
Register CL: 103
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Return Code
AH: Physical Error, or Record Count
BX: Same as AX

Function
The F_WRITEXFCB system call creates a new XFCB, or updates the existing XFCB for the specified file.

Entry conditions
The calling program sets the DR (drive) field, the F1 through F8 and T1 through T3 (filename and filetype) fields in the referenced FCB, and sets the EX field of the FCB to specify the password mode and whether a new password is to be assigned to the file. The following bits are defined for the EX field in the FCB:

Byte 12: Password Mode field
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 8 bytes of the current DMA buffer, or have been previously established as the default password (see the F_PASSWD (Function 106) system call). If bit 0 is set to 1, the new password must reside in the second 8 bytes of the current DMA buffer.

System action
The system creates a new XFCB, or updates the existing XFCB, for the specified file with the Password Mode as set in byte 12 of the referenced FCB. If requested, the system assigns a new password to the file.

If the F_WRITEXFCB operation is successful, the system sets registers AL and AH to 00h.

If no directory label exists for the specified drive, or the file specified is not found, or no more space exists in the directory for the XFCB, or passwords are not enabled for the drive, the system sets register AL to 0FFh, and register AH to 00h.

If a physical or extended error is encountered and the File System Error Mode is in default mode, the system displays a message identifying the error, and terminates. Otherwise, the system sets register AL to 0FFh, and register AH 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 the FCB filename or filetype field

BDOS Function 40: F_WRITEZF (Write Random with Zero Fill)

Entry Parameters:
Register CL: 40
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AL: Return Code
AH: Physical Error, or Record Count
BX: Same as AX

Function
The F_WRITEZF system call is identical to the F_WRITERAND (Function 34) system call, except that F_WRITEZF fills a previously unallocated data block with zeroes before writing the record. These zero-filled records identify unwritten random records. When using the F_WRITERAND (Function 34) system call, the unwritten random records in allocated data blocks contain uninitialized data.

List device I/O system calls

The list device I/O pertains to functions operating on the system list device LST:. Table 6-16 lists all the list device system calls discussed in this section.

Table 6-16. List device I/O system calls

Mnemonic	Function	Description
L_WRITE	5	Write a character to LST:
L_WRITEBLK	112	Write a block of characters to LST:

BDOS Function 5: L_WRITE (List Output)

Entry Parameters:
Register CL: 5
DL: Character

Function
The L_WRITE system call sends the character in DL to the list device LST:.

System action
The system sends the character in DL to the list device LST:.

If the device is not ready, the system waits for the device to be ready.

BDOS Function 112: L_WRITEBLK (List Block)

Entry Parameters:
Register CL: 112
DX: CHCB Address -- Offset
DS: CHCB Address -- Segment

Function
The L_WRITEBLK system call sends the character string referenced by the CHaracter Control Block (CHCB) to the list device LST:.

The CHCB format is the following:

Bytes 0 and 1: Offset of character string
Bytes 2 and 3: Segment of character string
Bytes 4 and 5: Length of character string (word value)

Memory management system calls

The memory management system calls allocate and free memory segments. Table 6-17 lists all the memory management system calls described in this section.

Table 6-17. Memory management system calls

Mnemonic	Function	Description
MC_ABSALLOC	56	Allocate absolute exact memory
MC_ABSMAX	54	Allocate absolute maximum memory
MC_ALLFREE	58	Free all memory
MC_ALLOC	55	Allocate exact amount of memory
MC_FREE	57	Free memory
MC_MAX	53	Allocate maximum memory

The MC_ memory management system calls use the Memory Control Block (MCB) to pass parameters to and from the operating system. The MCB is defined as follows:

```
MCB_Segment     EQU     WORD PTR 0      ; Start  of memory allocation
MCB_Length      EQU     WORD PTR 2      ; Length of memory allocation
MCB_Ext         EQU     BYTE PTR 4      ; Returned byte value,
                                        ;   defined with each function.
```

BDOS Function 56: MC_ABSALLOC (Allocate Absolute Exact Memory)

Entry Parameters:
Register CL: 56
DX: MCB Address -- Offset
DS: MCB Address -- Segment
MCB_Segment: Paragraph address
MCB_Length : Number of paragraphs to be allocated
MCB_Ext : 02h if region is to remain allocated

after program terminates.

Returned Values:
Register AX: 0000h = Successful, 0FFFFh = Not successful
BX: Same as AX
CX: Error Code (See Table 6-4)
MCB_Segment: Base of allocated memory
MCB_Length : Number of paragraphs allocated
MBC_Ext : 00h if no additional memory available
01h if additional memory available

Function
The MC_ABSALLOC system call allocates memory area at the address specified by MCB_Segment for the number of paragraphs specified by MCB_Length.

Entry conditions
The calling program sets MCB_Segment to the base address of the paragraph at which the allocated region is to start, and MCB_Length to the number of paragraphs to be allocated. The program also sets MCB_Ext to 02h if the region is to remain allocated after the program terminates.

System action
The system allocates a memory area that starts at the base address specified by MCB_Segment. The size of the memory area is specified by MCB_Length. The system sets MCB_Ext to 00h if no additional memory is available, or to 01h if additional memory is available.

If the allocation is successful, the system sets register AX to 0000h. Otherwise, it sets register AX to 0FFFFh.

BDOS Function 54: MC_ABSMAX (Allocate Absolute Maximum Memory)

Entry Parameters:
Register CL: 54
DX: MCB Address -- Offset
DS: MCB Address -- Segment
MCB_Segment: Paragraph address
MCB_Length : Number of paragraphs to allocate
MCB_Ext : 02h if region is to remain allocated
after program terminates.

Returned Values:
Register AX: 0000h = Successful, 0FFFFh = Not successful
BX: Same as AX
CX: Error Code (See Table 6-4)
MCB_Segment: Base of allocated memory
MCB_Length : Number of paragraphs allocated
MCB_Ext : 00h if no additional memory available
01h if additional memory available

Function
The MC_ABSMAX system call allocates the largest possible region at the absolute paragraph boundary specified by MCB_Segment for a maximum of

MCB_Length.

Entry conditions
The calling program sets MCB_Segment to the paragraph address at which allocation is to begin, and MCB_Length to the maximum number of paragraphs to be allocated. The program sets MCB_Ext to 02h if the region is to remain allocated after the program terminates.

System action
The system sets MCB_Segment to the base address of the allocated memory, and MCB_Length to the number of paragraphs allocated, and MCB_Ext to 00h if no additional memory is available, or to 01h if additional memory is available.

If the allocation is successful, the system sets register AX to 0000h. Otherwise, it sets AX to 0FFFFh.

BDOS Function 58: MC_ALLFREE (Free All Memory)

Entry Parameters:
Register CL: 58

Function
The MC_ALLFREE system call is included for compatibility only with CPM-86 Versions 1.0 and 1.1. You are referred to documentation provided for that system for description of this function. This function might not be supported in future releases of the operating system.

(ROCHE> The "documentation" is only the following sentence: "Function 58 is used to release all memory in the CP/M-86 environment (normally used only by the CCP upon initialization).")

BDOS Function 55: MC_ALLOC (Allocate Exact Amount of Memory)

Entry Parameters:
Register CL: 55
DX: MCB Address -- Offset
DS: MCB Address -- Segment
MCB_Length : Number of paragraphs to be allocated
MCB_Ext : 02h if region is to remain allocated
after program terminates.

Returned Values:
Register AX: 0000h = Successful, 0FFFFh = Not successful
BX: Same as AX
CX: Error Code (See Table 6-4)
MCB_Segment: Base of allocated memory
MCB_Length : Number of paragraphs allocated
MCB_Ext : 00h if no additional memory available
01h if additional memory available

Function
The MC_ALLOC system call allocates memory area specified by MCB_Length.

Entry conditions
The calling program sets MCB_Length to the number of paragraphs to be allocated, and sets MCB_Ext to 02h if the region is to remain allocated after the program terminates.

System action
The system sets MCB_Segment to the base address of the allocated memory, MCB_Length to the number of paragraphs allocated, and MCB_Ext to 00h if no additional memory is available, or to 01h if additional memory is available.

If the allocation is successful, the system sets register AX to 0000h. Otherwise, it sets AX to 0FFFFh.

BDOS Function 57: MC_FREE (Free Memory)

Entry Parameters:
Register CL: 57
DX: MCB Address -- Offset
DS: MCB Address -- Segment
MCB_Segment: Start of allocated memory
MCB_Length : Number of paragraphs allocated
MCB_Ext : 00h = Free this allocation
0FFh = Free all memory allocated to the program

Returned Values:
Register AX: 0000h = Successful, 0FFFFh = Not successful
BX: Same as AX
CX: Error Code (See Table 6-4)
MCB_Segment: Base of allocated memory
MCB_Length : Number of paragraphs allocated

Function
The MC_FREE system call frees the specified allocated memory to the end of the allocated segment that includes the specified paragraph.

Entry conditions
The calling program sets MCB_Segment to the paragraph address of the memory to be released.

System action
The system releases memory starting at the address specified in the MCB_Segment to the end of the previously allocated segment that contains the specified paragraph.

If the allocation is successful, the system sets register AX to 0000h. Otherwise, it sets AX to 0FFFFh.

Note: Either an entire allocated region must be released, or the end of a region can be released; the middle section of a region cannot be released.

BDOS Function 53: MC_MAX (Allocate Maximum Memory)

Entry Parameters:
Register CL: 53
DX: MCB Address -- Offset
DS: MCB Address -- Segment
MCB_Length : Number of paragraphs to be allocated
MCB_Ext : 02h if region is to remain allocated
after program terminates.

Returned Values:
Register AX: 0000h = Successful, 0FFFFh = Not successful
BX: Same as AX
CX: Error Code (See Table 6-4)
MCB_Segment: Base of allocated memory
MCB_Length : Number of paragraphs allocated
MCB_Ext : 00h if no additional memory available
01h if additional memory available

Function
The MC_MAX system call allocates the largest available memory area that is less than or equal to the length specified by MCB_Segment.

Entry conditions
The calling program sets MCB_Length to the number of paragraphs to be allocated. It also sets MCB_Ext to 02h if the region is to remain allocated after the program terminates.

System action
The system allocates the largest available memory area that is less than or equal to the length specified by MCB_Length.

The system sets MCB_Segment to the base address of the allocated memory region, and sets MCB_Ext to 00h if no additional memory is available, or to 01h if additional memory is available.

If the allocation is successful, the system sets register AX to 0000h. Otherwise, it sets AX to 0FFFFh.

Program execution system calls

All system calls described in this section pertain to program load and execution, and program termination. Table 6-18 summarizes the functions in this section.

Table 6-18. Program execution system calls

Mnemonic	Function	Description
P_CHAIN	47	Chain to next program
P_CODE	108	Set program return code
P_DELAY	141	Delay program
P_DISPATCH	142	Relinquish processor

P_LOAD 59 Load CMD or RSX module
P_RSX 60 Call Resident System Extension
P_TERM 143 Terminate calling program
P_TERMCPM 0 Terminate calling program

BDOS Function 47: P_CHAIN (Chain to Program)

Entry Parameters:
Register CL: 47
DMA Buffer: Command Line

Returned Values:
Register AX: 0FFFFh = No CMD file
BX: Same as AX
CX: Error Code (See Table 6-4)

Function
The P_CHAIN system call provide a means of chaining from one program to the next without operator intervention.

Entry conditions
The calling program places a command line terminated by a null byte (00h) in the default DMA buffer.

System action
The system searches for the file under the Current User Number and user 0 on the default drive. If the system does not find the file, it sets AX to 0FFFFh, and returns to the calling program.

If the system finds the file, it releases the calling program's memory area, and tries to load the file. If the memory required for the file is not available to load the chained program, the system terminates the calling program. If the program is running in the foreground, the system also displays a message on the console.

If the load operation is successful, the system transfers control to the program.

Note: The P_CODE system call can be used to pass a two-byte value to the chained program.

BDOS Function 108: P_CODE (Get/Set Program Return Code)

Entry Parameters:
Register CL: 108
DX: Program Return Code (Set), or 0FFFFh (Get)

Returned Values:
Register AX: Program Return Code (Get)
BX: Same as AX

Function

The P_CODE system call sets or returns the Program Return Code.

Entry conditions
The calling program sets register DX to 0FFFFh if the request is to get the Program Return Code. Otherwise, the request is to set the Program Return Code to the value in register DX.

System action
If register DX = 0FFFFh, the system returns the current Program Return Code in register AX. Otherwise, the system sets the Program Return Code to the value passed by the calling program in register DX.

The P_CODE system call provides a mechanism for programs to pass an error code or a value to a subsequent command. Program Return Codes can also be used by programs to pass an error code or value to a chained program (see P_CHAIN (Function 47) system call).

A program can set or interrogate the Program Return Code by using the P_CODE system call. If register DX = 0FFFFh, then the current Program Return Code is returned in register AX. Otherwise, P_CODE sets the Program Return Code to the value in register DX. Table 6-19 defines Program Return Codes.

Table 6-19. Program Return Codes

Code (Hex)	Meaning
0000-FEFF	Successful return
FF00-FFFE	Unsuccessful return
0000	The operating system initializes the Program Return Code to 0000h, unless the program is loaded as the result of a P_CHAIN (Function 47) system call from a previous transient program.
FF80-FFFC	Reserved
FFFD	The program is terminated because of a fatal system error.
FFFE	The program is terminated by the system because the user typed a Ctrl-C.

BDOS Function 141: P_DELAY (Delay Program for Specified ticks)

Entry Parameters:
Register CL: 141
DX: Number of ticks to delay

Returned Values:
Register AX: 0000h = Delay supported, 0FFFFh = Delay not supported
BX: Same as AX

Function
The P_DELAY system call provides a means of delaying a program and relinquishing the processor during the delay interval.

Entry conditions
The calling program specifies the number of system ticks. The number of ticks per second can be determined via an S_SYSVAR (Function 49) system call. (It is

normally 50 ticks per second in Europe, 60 in the USA.) Control is not returned to the calling program until the specified interval has occurred.

If the implementation of CP/M-86 Plus does not support the system tick, then P_DELAY returns an error.

(ROCHE> The "Concurrent CP/M 3.1 Programmer's Guide" gives the following explanation:

The P_DELAY system call causes the calling process to wait until the specified number of system ticks has occurred. The P_DELAY system call avoids the necessity of programmed delay loops. It allows other processes to use the CPU resource, while the calling process waits.

The length of the system tick varies among installations. A typical system tick is 60 Hz (16.67 milliseconds) in the USA. In Europe, it is likely to be 50 Hz (20 milliseconds). The exact length of the system tick can be obtained by reading the TICKS/SEC value from the System Data Segment (refer to the S_SYSDAT system call).

There is up to one tick of uncertainty in the exact amount of time delayed. This is due to the P_DELAY system call being called asynchronously from the actual time base. The P_DELAY system call is guaranteed to delay the calling process at least the number of ticks specified. However, when the calling process is rescheduled to run, it might wait quite a bit longer if there are higher priority processes waiting to run. The P_DELAY system call is used primarily by programs that need to wait specific amounts of time for I/O events to occur. Under these conditions, the calling process usually has a very high priority level. If a process with a high priority calls the P_DELAY system call, the actual delay is typically within a system tick of the amount of time wanted.)

BDOS Function 142: P_DISPATCH (Relinquish Processor)

Entry Parameters:
Register CL: 142

Returned Values:
Register AX: 0000h = Control returned Ok, 0FFFFh = Dispatch not supported

Function
The P_DISPATCH system call provides a means of relinquishing the processor to other running programs. All running programs are given the appropriate time ratio (set by the CP/M-86 Plus built-in command FORE) before control is returned to the calling program.

BDOS Function 59: P_LOAD (Load CMD or RSX)

Entry Parameters:
Register CL: 59
DX: FCB Address -- Offset
DS: FCB Address -- Segment

Returned Values:
Register AX: Base Page address (0FFFFh = Error)
BX: Same as AX
CX: Error Code (See Table 6-4)

Function
The P_LOAD system call loads a CMD file or an RSX file into memory.

Entry conditions
The calling program must have successfully opened the CMD or RSX file specified in the referenced FCB.

System action
The system loads the CMD module, and sets both registers AX and BX to the Base Page address for the loaded program. It does not call the loaded program.

If the file Header Record indicates an RSX only, the system loads the RSX module, and sets both registers AX and BX to 0000h.

If the memory required to load the program is not available, or if an error occurs while reading the file, the system sets register AX to 0FFFFh, and returns.

BDOS Function 60: P_RSX (Call Resident System Extension)

Entry Parameters:
Register CL: 60
DX: RSX PB Address

Returned Values:
Register AX: 0FFFFh = No RSX in memory

Function
The P_RSX system call is a special system function to use when calling Resident System Extensions.

Entry conditions
The RSX subfunction is specified in a structure called the RSX Parameter Block, defined as follows:

```
RSXPB   DB      FUNC            ; RSX Subfunction Number
        DB      NumParms        ; Number of word parameters
        DW      Parm1           ; Parameter 1
        DW      Parm2           ; Parameter 2
        ..      ...
        DW      ParmN           ; Parameter N
```

System action
RSX modules in memory filter all system calls, and intercepts RSX function calls that they can handle. The program uses the P_RSX system call and an associated subfunction number to call an RSX. The RSX module that handles the specific function call must be in memory.

If the RSX module is not present, the call is not intercepted, and the system returns a 0FFFFh in register AX. RSX subfunction numbers from 0 to 127 are available for CP/M-compatible software use. RSX subfunction numbers from 128 to 255 are reserved for system use.

BDOS Function 0 or 143: P_TERMCPM or P_TERM (Terminate Program)

Entry Parameters:
Register CL: 0, or 143
DL: Abort Code

Function
The P_TERMCPM or P_TERM system calls terminate the calling program, and return control to the system.

Entry condition
The calling program sets register DL to 00h if all the memory belonging to the program is to be released. Otherwise, the calling program sets register DL to 01h if the program and its buffers are to remain in memory.

Note: The program can set the Program Return Code by making a P_CODE (Function 108) system call prior to making a P_TERMCPM or P_TERM system call.

System action
The system reloads the CCP, if necessary, rebuilds the allocation vectors for the currently logged-in drives, sets the DMA buffer offset to 0080h, and transfers control to the CCP. Furthermore, if DL = 00h, the system releases the memory and buffers of the calling program. If DL = 01h, the program remains in memory, and the memory allocation state remains unchanged.

Miscellaneous system calls

This section includes calls to set the system values and parameters, the direct call to the BIOS, and time-related system calls. Table 6-20 lists all the system calls described in this section.

Table 6-20. Miscellaneous system calls

Mnemonic	Function	Description
S_BDOSVER	12	Return BDOS Version Number
S_BIOS	50	Direct BIOS calls
S_SERIAL	107	Return Serial Number
S_SYSDAT	154	Get System Data address
S_SYSVAR	49	Get/Set system variables
T_GET	105	Get date and time
T_SET	104	Set date and time

BDOS Function 12: S_BDOSVER (Return BDOS Version Number)

Entry Parameters:
Register CL: 12

Returned Values:
Register AL: 31 (BDOS Version Number: 3 and Revision Level: 1)
AH: 10 (8086 CP/M)
BX: Same as AX

Function
The S_BDOSVER system call returns a two-byte value containing the BDOS Version Number and Release Level, and whether the system is CP/M or MP/M, and 8-bit or 16-bit.

System action
ROCHE> The "Concurrent CP/M 3.1 Programmer's Guide" gives the following explanation:

The S_BDOSVER system call returns the BDOS file system Version Number, allowing version-independent programming.

AL High Nibble = BDOS Version Number
AL Low Nibble = BDOS Revision Level
AH High Nibble = CPU Type : 0 = 8080, 1 = 8086
AH Low Nibble = OS Type : 0 = CP/M
1 = MP/M
2 = CP/M with networking
3 = MP/M with networking
4 = Concurrent CP/M
5 = Reserved
6 = Concurrent CP/M with networking
7 to 0Eh = Reserved

BDOS Function 50: S_BIOS (Direct BIOS Call)

Entry Parameters:
Register CL: 50
DX: BIOS PB Address -- Offset
DS: BIOS PB Address -- Segment

Returned Values:
Register AX: BIOS Return Code
BX: Same as AX

Function
The S_BIOS system call provides a direct BIOS call, and transfers control to the BIOS through the BDOS.

Entry condition
The DX register addresses a 5-byte memory area containing the BIOS Parameter Block:

```
BIOSPB  DB      FUNC            ; BIOS function number
```

```
    DW      CX_reg      ; CX register contents
    DW      DX_reg      ; DX register contents
```

FUNC is an 8-bit BIOS function number, and values CX_REG and DX_REG are the 16-bit values that would normally be passed directly in the CX and DX registers with the BIOS call. The CX and DX values are loaded into the 8086 registers before the BIOS call is initiated. See the "CP/M-86 Plus System Guide" for descriptions of the BIOS functions.

The only allowed BIOS function numbers under CP/M-86 Plus are the following:

Table 6-21. BIOS functions allowed with CP/M-86 Plus

Func.	Description
0	Console status
1	Console input
2	Console output
3	List output status
4	List output
5	Auxiliary input
6	Auxiliary output
7	Not implemented (Concurrent CP/M function)
8	Not implemented (Concurrent CP/M function)
9	Select disk
10	Read sector
11	Write sector
12	Not implemented (CP/M-86 function)
13	Not implemented (Concurrent CP/M function)
14	Character device initialization
15	Console output status
16	Auxiliary input status
17	Auxiliary output status

BDOS Function 107: S_SERIAL (Return Serial Number)

Entry Parameters:
Register CL: 107
DX: SERIAL Address -- Offset
DS: SERIAL Address -- Segment

Returned Values:
SERIAL number filled in

Function
The S_SERIAL system call sets a 6-byte structure SERIAL to the CP/M-86 Plus serial number.

System action
The system sets the serial number in the SERIAL data structure.

Each byte in the structure contains an ASCII number.

BDOS Function 154: S_SYSDAT (Get System Data Address)

Entry Parameters:
Register CL: 154

Returned Values:
Register BX: SYSDAT Address -- Offset
ES: SYSDAT Address -- Segment

Function
The S_SYSDAT system call returns the segment address of the System Data Area. This call provides the ability for advanced programmers to access internal data and structures not accessible via the "normal" BDOS calls.

These are documented in "Appendix C: SYSDAT format" of the "CP/M-86 Plus System Guide", but programmers are reminded that, as these locations may differ in future version of CP/M-86 Plus, they should be used with discretion.

BDOS Function 49: S_SYSVAR (Get/Set System Variables)

Entry Parameters:
Register CL: 49
DX: SCB PB Address -- Offset
DS: SCB PB Address -- Segment

Returned Values:
SCB PB filled in if Get

Function
The S_SYSVAR system call allows access to system variables that do not have specific system calls associated with them.

Entry conditions
A structure called the SCB Parameter Block (SCBPB) is used to define the calling and return parameters. The SCBPB is defined as follows:

```
SCBPB   DB      Num             ; Number identifying the variable
        DB      Set             ; 0FFh = Set, 00h = Get
        RS      Value           ; VALUE ranges from 1 to 5 bytes,
                                ;   depending on the variable.
```

The system variable numbers to be set in NUM are defined as follows:

Table 6-22. SCBPB variable numbers

Number	Definition
0	Console Width This parameter specifies the number of columns, or characters per line, on the console. This byte value is relative to zero, and defaults to 79 on most systems. The Console Width can be set by the DEVICE command (see the "CP/M-86 PLus User's Guide"). Note that typing

a character into the last position specified by the Console Width field must not cause the cursor to advance to the next line position.

1 Console Page Length
This parameter defines the page length (lines per page) of the console. This byte value defaults to 24 on most systems. The DEVICE command can be used to change this value (see the "CP/M-86 Plus User's Guide").

2 Console Page Mode
If this byte is set to 1, the system displays one full screen of information at a time. If this byte is set to 0, the system displays information continuously, scrolling after the screen is full. Display is interrupted and continued by the Ctrl-S, Ctrl-Q sequence. The SETDEF command can be used to change this value (see the "CP/M-86 Plus User's Guide").

3 System Ticks per Second
This byte value contains the number of ticks per second of the system clock. The program can only request this value, not set it.

4 Temporary File Drive
This byte value contains the drive number of the temporary file drive. The drive number ranges from 0 through 16, where 0 corresponds to the default drive, while 1 through 16 correspond to drives A through P, respectively. The SETDEF command can be used to change this value (see the "CP/M-86 Plus User's Guide").

5 Date and Time
The 5-byte time and date information is set in the data block starting at the reserved area whose structure is similar to the DAT structure (see the T_GET (Function 105) system call). The DATE command can be used to change these values (see the "CP/M-86 Plus User's Guide").

System action
If the SET parameter is set to 0FFh, the system updates the system variable defined by NUM to the value assigned by the program. Otherwise, the system returns the requested system information in the reserved area.

(ROCHE> One of my program displays the following:

System Variables

Console Width: 80
Console Page Length: 24
Console Page Mode: OFF
System Ticks per Second: 60
Temporary File Drive: @
Date: 04DC
Time: 12:34:56
8087 Present: TRUE
Program ID: 00
Drive Search Chain: A, B, C, D

In the Background: FALSE
Number of Running Processes: 01
Foreground Ratio: 16

So, some fields are not documented...)

BDOS Function 105: T_GET (Get Date and Time)

Entry Parameters:
Register CL: 105
DX: DAT Address -- Offset
DS: DAT Address -- Segment

Returned Values:
Register AL: Seconds
BL: Same as AL
DAT filled in (Days, Hours, and Minutes only)

Function
The T_GET system call returns the system internal date and time.

Entry conditions
The DX register is set to the address of a 4-byte data structure to contain the date and time values. The number of seconds is returned in register AL as a two-digit BCD value. The format of the DAT structure is as follows:

```
DAT     DW      0000h   ; Date field
        DB      00h     ; Hour field
        DB      00h     ; 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.

System action
The system places the date and time information in the referenced DAT area. The seconds field is returned in register AL.

BDOS Function 104: T_SET (Set Date and Time)

Entry Parameters:
Register CL: 104
DX: DAT Address -- Offset
DS: DAT Address -- Segment

Function
The T_SET system call sets the system internal date and time to the date and time specified in the DAT structure, as described in the T_GET (Function 105) system call.

Entry conditions
The DX register is set to the address of a 4-byte data structure containing

the date and time values. The format of the DAT structure is as follows:

```
DAT     DW      0000h   ; Date field
        DB      00h     ; Hour field
        DB      00h     ; 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.

System action
The system sets the internal date and time to the values contained in the referenced DAT. The seconds field is set to zero.

EOF

CPMPPG7.WS4 (= "CP/M-86 Plus Programmer's Guide", section 7)

(Edited by Emmanuel ROCHE.)

Section 7: Resident System Extensions (RSXs)

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 5 for a discussion of how RSXs operate under CP/M-86 Plus.) 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.

RSX prefix

The first 32 bytes of an RSX program contain a standard data structure called the RSX prefix. The RSX prefix has the following format:

```
        CSEG
start:  JMP     ftest           ; Jump to start of program
;
term    DB      0FFh            ; Terminate flag: Set to 0FFh to remove
                                ;   this RSX after program terminates.
                                ;   Set to 00h if RSX stays in memory.
next    DW      0,0             ; Pointer to next RSX in chain
rname   DB      'ECHOVERS'      ; Name of this RSX
RSXDS   RS      2               ; Data Segment of this RSX
RSXID   RS      1               ; Program ID of this RSX
RSXnd   RS      2               ; (Not documented)
        RS      11              ; Reserved
;
ftest:  ...                     ; BDOS Function test
        ...
        CALLF DWORD PTR next    ; BDOS Call (replaces INT 224)
        ...
        RETF                    ; Return to CMD file
```

The only fields of the RSX prefix that you must initialize are the TERM flag and the RNAME of the RSX.

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 (FTEST means "(BDOS) Function Test".)

The NEXT field contains the Offset and Segment address of the next RSX module in the chain, or the address of the LOADER program 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 TERM field controls whether the RSX is removed from memory by the next call to P_LOAD (Function 59). If the TERM flag is 0FFh, the system removes the RSX from memory when the program terminates.

Example of RSX use

These 2 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 (S_BDOSVER) call to obtain the CP/M-86 Plus 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 Number 1031h to CALLVERS.

Although this example is simple, it illustrates BDOS function interception, stack swapping, and BDOS function calls within an RSX.

Listing 7-1. Sample program using RSX

```
; CALLVERS.A86
; ------------
;
; CP/M-86 Plus -- CALLVERS.CMD
;
; CALLVERS prints a message to the console, then makes a BDOS
; Function 12 call to obtain the BDOS Version Number. CALLVERS
; repeats this sequence 5 times before terminating.
;
;--------------------------------
;
bdos    EQU     224             ; CP/M-86 Plus's BDOS
;
; BDOS functions used.
;
P_TERMCPM       EQU     0       ; System Reset
C_WRITESTR      EQU     9       ; Print String
S_BDOSVER       EQU     12      ; BDOS Version Number
;
; ASCII characters used.
;
lf      EQU     0Ah             ; Line Feed
cr      EQU     0Dh             ; Carriage Return
;
;--------------------------------
; Small Memory model.
;
        CSEG
```

```
        DSEG
        ORG     0100h
;
CallMsg DB      cr, lf, '**** CALLVERS **** $'
;
;--------------------------------
        CSEG    $
;--------------------------------
;
        MOV     DL, 5           ; Perform loop 5 times
Loop:   PUSH    DX              ; Save loop counter
        MOV     CL, C_WRITESTR  ; Print **** CALLVERS ****
        MOV     DX, OFFSET CallMsg
        INT     bdos            ; Go through the RSX chain
        MOV     CL, S_BDOSVER   ; Get BDOS Version Number
        INT     bdos            ; Go through the RSX chain
        POP     DX              ; Restore loop counter
        DEC     DL              ; Decrement loop counter
        JNZ     Loop            ; If counter > 0, then loop through again
;
        MOV     CL, P_TERMCPM   ; Warm Boot
        MOV     DL, CL          ; Remove RSX from memory
        INT     bdos            ; Go through RSX chain
;
;--------------------------------
;
        END
```

Listing 7-2. Sample program of an RSX

```
; ECHOVERS.A86
; ------------
;
; CP/M-86 Plus -- ECHOVERS.RSX
;
; ECHOVERS is an RSX that intercepts the BDOS Function 12 call
; made by CALLVERS, prints a second message, and returns the
; CP/M-86 Plus Version Number 1031h to CALLVERS.
;
; Although this example is simple, it illustrates BDOS function
; interception, stack swapping, and BDOS function calls within an RSX.
;
;--------------------------------
; BDOS functions used.
;
C_WRITESTR      EQU     9       ; Print String
S_BDOSVER       EQU     12      ; BDOS Version Number
;
; ASCII characters used.
;
lf      EQU     0Ah             ; Line Feed
cr      EQU     0Dh             ; Carriage Return
;
;--------------------------------
```

```
; Small Memory model.
;
        CSEG
        DSEG
        ORG     0100h
;
EchoMsg DB      cr, lf, '**** ECHOVERS **** $'
;
Ret_ss  DW      0000h           ; Stack Segment save area
Ret_sp  DW      0000h           ; Stack Pointer save area
Stack   RW      0100h           ; 128-word stack
Loc_Stk RS      0               ; Beginning of RSX stack
;
;--------------------------------
        CSEG    $
;--------------------------------
; RSX prefix.
;
        JMP     Ftest           ; Go test BDOS Function number
;
        DB      0FFh            ; Terminate RSX flag = TRUE
next    DW      0,0             ; Pointer to next RSX in chain
rname   DB      'ECHOVERS'      ; Name of this RSX
RSXDS   DW      0000h           ; Data Segment of this RSX
RSXID   DB      00h             ; Program ID of this RSX
RSXnd   DW      0000h           ; (Not documented)
        RS      11              ; Reserved area
;
;--------------------------------
; Start of RSX code.
;
Ftest:
;
; Normally, the purpose of an RSX is to intercept BDOS function calls.
; So, the first thing to do is to check the BDOS Function number.
;
        MOV     AL, CL          ; Put BDOS Function number in Accumulator
        CMP     AL, S_BDOSVER   ; Is it BDOS Function 12 ?
        JZ      Begin           ; Yes: Intercept this call
        JMPF    DWORD PTR next  ; No: Some other BDOS Function call
;
;--------------------------------
Begin:
;
; Like all Small Memory Model program, this RSX has distinct
; Code and Data segments (CS and DS).
; The problem is that, if we want to print a string contained
; in the RSX, the DS must be updated, else we will print some
; garbage from the CMD file...
;
        PUSH    DS              ; Save calling CMD file's DS
        MOV     DX, RSXDS       ; Get DS from the RSX's Prefix
        MOV     DS, DX          ; DS updated
;
```

```
; Last thing to do, before processing: swap the stack to the local stack
; (since the stack is also in the calling CMD file's DS...).
;
        MOV     Ret_ss, SS      ; Save Stack Segment
        MOV     Ret_sp, SP      ; Save Stack Pointer
        MOV     AX, DS          ;
        MOV     SS, AX          ; Switch to local stack
        MOV     SP, OFFSET Loc_Stk
;
; RSX Initialization done.
;-------------------------------
; We can finally print a second message.
;
        MOV     CL, C_WRITESTR  ; Print **** ECHOVERS ****
        MOV     DX, OFFSET EchoMsg
;
; Note that the standard "INT 224" is replaced by the following:
;
        CALLF   DWORD PTR next  ; Call the BDOS through the RSX chain
;
; And return the BDOS Version Number 1031h to CALLVERS.
; (1031h = CP/M-86 Plus)
; (1131h = MP/M-86 with BDOS 3.1)
: (1431h = Concurrent CP/M 3.1)
;
        MOV     BX, 1031h       ; Return BDOS 'Version 3.1'
;
; Done.
;-------------------------------
; Standard end of RSX.
;
        MOV     SS, Ret_ss      ; Restore Stack Segment
        MOV     SP, Ret_sp      ; Restore Stack Pointer
        POP     DS              ; Restore CMD file's DS
        RETF                    ; Return to calling CMD file
;
;-------------------------------
;
        END
```

You can prepare the preceding programs for execution as follows:

1) Assemble the CALLVERS program using ASM-86:

```
A>ASM CALLVERS
```

2) Generate a CMD file for CALLVERS with GENCMD:

```
A>GENCMD CALLVERS
```

3) Assemble the RSX program ECHOVERS using ASM-86:

```
A>ASM ECHOVERS
```

4) Generate a CMD file using the GENCMD command:

```
A>GENCMD ECHOVERS
```

5) Convert ECHOVERS.CMD to an RSX file:

```
A>GENRSX ECHOVERS
```

6) Attach the RSX file to CALLVERS by using the GENRSX utility:

```
A>GENRSX CALLVERS ECHOVERS [ATTACH]
```

7) Run the CALLVERS.CMD module:

```
A>CALLVERS
```

The message

```
**** CALLVERS ****
```

followed by the message

```
**** ECHOVERS ****
```

appears on the screen five times if the RSX program works.

The GENRSX utility

The GENRSX utility creates an RSX file from a CMD file, or modifies a CMD file to link or attach RSX programs to the CMD. Conversely, the GENRSX utility can strip linked or attached RSXs from a CMD file.

The GENRSX command has the following format:

```
GENRSX CMDfilespec {RSXfilespec, ..., RSXfilespec} [ATTACH/STRIP]
```

These elements are as follows:

- CMDfilespec is a file of type CMD
- RSXfilespec is a file of type RSX
- ATTACH specifies that the RSX files are to be attached to the CMD file
- STRIP specifies that the RSX files are to be removed from the CMD file

GENRSX functions

GENRSX performs the following functions:

1) Converts a CMD file to an RSX format.
2) Links RSX files to a CMD file.
3) Attaches RSX files to a CMD file.
4) Removes RSX files from a CMD file.

1) Convert CMD to RSX

Syntax: GENRSX CMDfilespec

Example: A>GENRSX A

The preceding example converts a CMD file to an RSX file. GENRSX searches for file A.CMD, and changes the file's Header Record to indicate that this is an RSX file. Then, it renames A.CMD to A.RSX.

Note: A valid RSX file cannot be generated just by renaming a CMD file to an RSX filetype. It must be processed by GENRSX.

An RSX file can be loaded into memory by entering just the filename and RSX filetype at the console, as follows:

A>A.RSX

While it is in memory, A.RSX intercepts each BDOS system call, and checks it to see if it should process this call, or pass it to the BDOS for normal processing.

2) Link RSX to CMD

Syntax: GENRSX CMDfilespec RSXfilespec ... RSXfilespec

Example: A>GENRSX A B C

This command links B.RSX and C.RSX to A.CMD. GENRSX searches for file A.CMD, B.RSX, and C.RSX. If it finds all the files, it modifies the CMD file Header Record and the RSX files Header Records to indicate that the CMD file has linked RSXs. When loading A.CMD, the system can tell from the Header Records which RSXs are to be loaded with the CMD file.

Each BDOS system call invoked by program A will be intercepted first by RSX C. If this call applies to C, then C processes the call and returns to program A. Otherwise, RSX C passes the call to RSX B. RSX B likewise checks the call for processing. If this call applies to B, then B processes it and returns to the program A. Otherwise, B passes the call to BDOS for normal processing.

3) Attach RSX to CMD

Syntax: GENRSX CMDfilespec RSXfilespec ... RSXfilespec [ATTACH]

Example: A>GENRSX A B C [ATTACH]

The above example attaches RSX files B.RSX and C.RSX to CMD file A.CMD. GENRSX

modifies the CMD file Header Record and the RSX file Headers Records, and generates a new file A.CMD.

The BDOS system calls invoked by program A are intercepted by RSXs C and B as described in the previous section.

Note: A command file can have both linked and attached RSXs. This process requires 2 GENRSX commands. For example, the following commands generate file A.CMD with linked RSXs B and C, and attached RSXs D and E:

```
A>GENRSX A B C
A>GENRSX A D E [ATTACH]
```

4) Remove RSX from CMD

Syntax: GENRSX CMDfilespec RSXfilespec ... RSXfilespec [STRIP]

Example: A>GENRSX A B C [STRIP]

The above example strippes RSX files B.RSX and C.RSX from CMD file A.CMD. GENRSX modifies the CMD file Header Record and the RSX file Headers Records, and generates a new file A.CMD.

The BDOS system calls invoked by program A are no longer intercepted by RSXs C and B.

EOF

CPMPPG8.WS4 (= "CP/M-86 Plus Programmer's Guide", section 8)

(Edited by Emmanuel ROCHE.)

Section 8: Escape sequences

CP/M-86 Plus allows you to use the following escape sequences in order to change the position or shape of the cursor on screen. These sequences can be divided into two groups: VT-52 sequences, and VT-100 sequences.

VT-52 sequences

ESC A -- Move cursor one line up
If the cursor was in the first line, the entire screen is moved one line downwards. The last line is lost. The new (first) line consists of blanks.

Example:
The cursor is in the line 1, column 10. ESC A is pressed. The screen is moved one line downwards, a blank line is inserted, and the cursor remains in the line 1, column 10.

ESC B -- Move cursor one line down
The cursor is moved one line downwards.

ESC C -- Move cursor forward one column
The cursor is moved one column to the right. If the cursor is in the last column, it is moved to the beginning of the next line.

ESC D -- Move cursor backward one column
The cursor is moved one column to the left. If the cursor is in the first column, it is moved to the end of the previous line.

ESC H -- Move cursor to HOME position
The cursor is positioned in the topmost left corner of the screen.

ESC Y<line><column> -- Position cursor

Parameter: <line> = line number (1 through 24)
<column> = column number (1 through 80)

Note: <line> = 0 or <column> = 0 is treated as 1.

Example: ESC Y!A
The cursor is positioned at line 1, column 21. Since this sequence does not require a conversion, the parameters are considered ASCII characters - 20h.

Example of ASCII characters and their results:

ASCII	Significance
!	= 21h - 20h = 1 decimal
2	= 32h - 20h = 12 decimal
M	= 4Dh - 20h = 45 decimal
p	= 70h - 20h = 80 decimal

ESC e -- Make cursor visible

ESC f -- Make cursor invisible

ESC j -- Save cursor position

ESC k -- Restore cursor position

ESC E -- Clear screen

ESC J -- Delete screen from current cursor position

ESC d -- Delete screen until current cursor position

ESC K -- Delete from cursor position to end of line

ESC o -- Delete from beginning of line until cursor position

ESC l -- Delete line

ESC L -- Insert line

ESC N -- Delete character

ESC M -- Remove line

ESC b<parameter> -- Set foreground color

ESC c<parameter> -- Set background color

The following <parameter>s apply to ESC b and ESC c:

0 - black	8 - grey
1 - blue	9 - light blue
2 - green	10 - light green
3 - cyan	11 - light cyan
4 - red	12 - light red
5 - magenta	13 - light magenta
6 - yellow	14 - light yellow
7 - white	15 - light white

These values must be entered as hexadecimal numbers, i.e. the function cannot be used via the keyboard. To execute the function, you need to write a small program.

ESC p -- Reverse video ON

ESC q -- Reverse video OFF

ESC r -- Intensity ON (high intensity)

ESC u -- Intensity OFF (low intensity)

ESC s -- Flashing ON

ESC t -- Flashing OFF

ESC v -- Automatical setting of CR/LF at end of line

ESC w -- No CR/LF at end of line

ESC z -- Set all attributes to normal

Normal condition means:
Black characters on white background,
Normal intensity,
Reverse OFF,
Flashing OFF,
Cursor visible.

ESC x -- Switch over to color card

ESC y -- Switch over to monochrome card

ESC 7 -- Function key expansion ON

ESC 6 -- Function key expansion OFF

VT-100 sequences

The following VT-100 escape control sequences have been implemented:

ESC [<line>;<column>H -- Position cursor

Parameter: <line> = line number (1 through 24)
<column> = column number (1 through 80)

Note: <line> = 0 or <column> = 0 is treated as 1.

Example: ESC[10;40H
The cursor is positioned at line 10, column 40.

ESC [<par>m -- Set screen control attributes

Parameter: <par> = 0 : Normal (See note (1) below)
<par> = 1 : Normal (See note (2) below)
<par> = 2 : Low intensity ON
<par> = 3 : Low intensity OFF

<par> = 4 : Normal (See note (1) below)
<par> = 5 : Flashing ON
<par> = 6 : Flashing OFF
<par> = 7 : Reverse video ON
<par> = 8 : Reverse video OFF
<par> = 9 : Insert CR,LF at end of line
<par> = 10 : No CR,LF at end of line

Notes:
(1) All values identical to those in ESC z, except for the cursor which remains invisible.
(2) All values identical to those in ESC z.

Example: ESC[7m
Following this sequence, all characters are reversed.

ESC [<par>c -- Set cursor mode

Parameter: <par> = 0 : Block, static
<par> = 1 : Underscore, static
<par> = 2 : Block, flashing
<par> = 3 : Underscore, flashing
<par> = 4 : Cursor, invisible
<par> = 5 : Block, slowly flashing
<par> = 6 : Underscore, slowly flashing

ESC [<par>K -- Delete line

Parameter: <par> = 0 : Delete from cursor position
<par> = 1 : Delete until cursor position
<par> = 2 : Delete complete line

Note:
The control characters do not change the position of the cursor.

Example:
The cursor is in line 10, column 7. Following ESC[0K, line 10 is deleted from column 7 onwards until the end of the line.

ESC [<par>J -- Clear screen

Parameter: <par> = 0 : Delete from cursor position
<par> = 1 : Delete until cursor position
<par> = 2 : Delete complete screen

Note:
The control characters do not change the position of the cursor.

Example:
The cursor is in line 9, column 11. Following ESC[0J, the screen is deleted from the cursor position until the end of the screen.

EOF