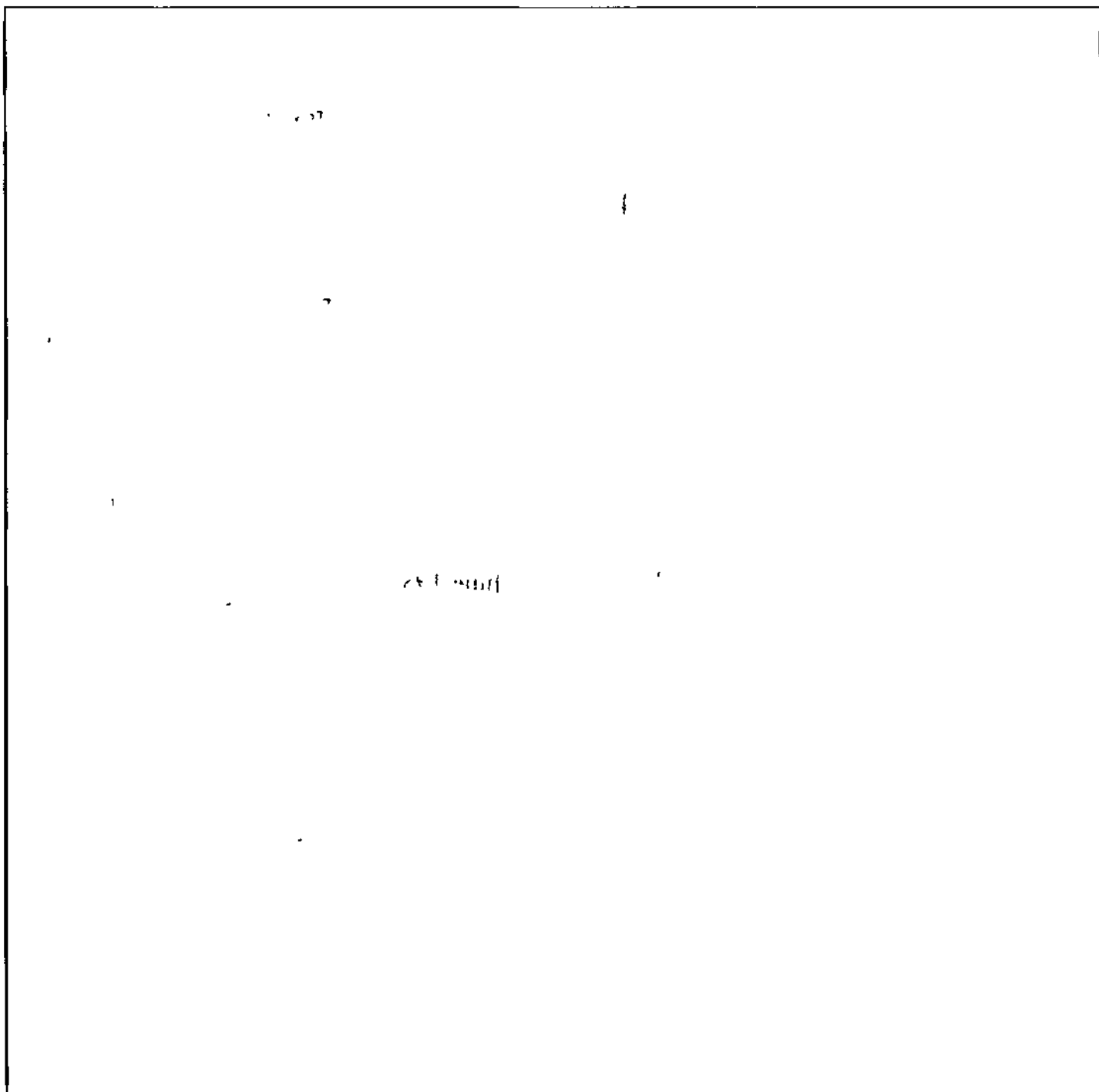


**SIEMENS**

**Betriebssystem  
CONCURRENT CP/M-86<sup>®</sup>**

**Programmer's  
Reference Guide**



## **COPYRIGHT**

Copyright © 1983 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California, 93950.

## **DISCLAIMER**

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

## **TRADEMARKS**

CP/M and CP/M-86 are registered trademarks of Digital Research. ASM-86, Concurrent CP/M-86, DDT, DDT-86, MP/M, and MP/M-86 are trademarks of Digital Research. Intel and MCS are registered trademarks of Intel Corporation. ISIS-II is a trademark of Intel Corporation. IBM is a registered trademark of International Business Machines.

*The Concurrent CP/M-86 Operating System Programmer's Reference Guide* was printed in the United States of America.

First Edition: June 1983

# Foreword

Concurrent CP/M-86™ is an operating system for 8086- or 8088-based microcomputer systems. It supports multiple CP/M® programming environments each implemented upon a virtual console. A different task runs concurrently in each environment.

This manual describes the invariant programming interface to Concurrent CP/M-86. It supports the applications programmer who must create applications programs that run in the CP/M-86® environment.

Section 1 offers an overview of the entire operating system.

Section 2 describes the structure of the Concurrent CP/M-86 file system.

Section 3 explains the format, structure, and uses of transient commands in the Concurrent CP/M-86 environment.

Section 4 explains the creation of transient command files in the Concurrent CP/M-86 environment.

Section 5 documents the structure and creation of resident system processes or resident command files permanently installed in the Concurrent CP/M-86 environment.

Section 6 describes all the Concurrent CP/M-86 system calls.

Concurrent CP/M-86 is supported and documented through four manuals:

- The *Concurrent CP/M-86 Operating System User's Guide* (hereafter cited as *Concurrent CP/M-86 User's Guide*) documents the user's interface to Concurrent CP/M-86, explaining the various features used to execute applications programs and Digital Research utility programs.

- The *Concurrent CP/M-86 Operating System Programmer's Reference Guide* (hereafter cited as *Concurrent CP/M-86 Programmer's Reference Guide*) documents the applications programmer's interface to Concurrent CP/M-86, explaining the internal file structure and system entry points, information that is essential for creating applications programs that run in the Concurrent CP/M-86 environment.
- The *Concurrent CP/M-86 Operating System Programmer's Utilities Guide* (hereafter cited as *Programmer's Utilities Guide*) is the Digital Research utility programs that programmers use to write, debug, and verify applications programs written for the Concurrent CP/M-86 environment.
- The *Concurrent CP/M-86 Operating System System Guide* (hereafter cited as *Concurrent CP/M-86 System Guide*) documents the internal, hardware-dependent structures of Concurrent CP/M-86.

# Table of Contents

## 1 Concurrent CP/M-86 System Overview

1.1	Introduction .....	1-1
1.2	Supervisor (SUP) .....	1-5
1.3	Real-time Monitor (RTM) .....	1-5
1.3.1	Process Dispatching .....	1-5
1.3.2	Queue Management .....	1-8
1.3.3	System Timing Function .....	1-9
1.4	Memory Module (MEM) .....	1-10
1.5	Basic Disk Operating System (BDOS) .....	1-11
1.6	Character I/O Module (CIO) .....	1-12
1.7	Virtual Console Screen Management .....	1-12
1.8	Extended Input/Output System (XIOS) .....	1-13
1.9	Terminal Message Processes (TMP) .....	1-14
1.10	Transient Programs .....	1-14
1.11	System Call Calling Conventions .....	1-15
1.12	SYSTAT: System Status .....	1-16

## 2 The Concurrent CP/M-86 File System

2.1	File System Overview .....	2-1
2.1.1	File-access System Calls .....	2-2
2.1.2	Drive-related System Calls .....	2-3
2.2	File Naming Conventions .....	2-5
2.3	Disk Drive and File Organization .....	2-8
2.4	File Control Block Definition .....	2-10
2.4.1	FCB Initialization and Usage .....	2-13
2.4.2	File Attributes .....	2-15
2.4.3	Interface Attributes .....	2-17
2.5	User Number Conventions .....	2-18
2.6	Directory Labels and XFCBs .....	2-19
2.7	File Passwords .....	2-23
2.8	File Date and Time Stamps: SFCBs .....	2-24
2.9	File Open Modes .....	2-27
2.10	File Security .....	2-29
2.11	Extended File Locking .....	2-32
2.12	Compatibility Attributes .....	2-34
2.13	Multisector I/O .....	2-37

# Table of Contents (continued)

2.14	Concurrent File Access .....	2-38
2.15	File Byte Counts .....	2-41
2.16	Record Blocking and Deblocking .....	2-42
2.17	Reset, Access, and Free Drive .....	2-43
2.18	BDOS Error Handling .....	2-47
<b>3</b>	<b>Transient Commands</b>	
3.1	Transient Process Load and Exit .....	3-1
3.2	Command File Format .....	3-2
3.3	Base Page Initialization .....	3-4
3.4	Parent/Child Relationships .....	3-7
<b>4</b>	<b>Command File Generation</b>	
4.1	Transient Execution Models .....	4-1
4.1.1	The 8080 Memory Model .....	4-2
4.1.2	The Small Memory Model .....	4-4
4.1.3	The Compact Memory Model .....	4-6
4.2	GENCMD .....	4-8
4.3	Intel Hexadecimal File Format .....	4-11
<b>5</b>	<b>Resident System Process Generation</b>	
5.1	Introduction to RSPs .....	5-1
5.2	RSP Memory Models .....	5-1
5.2.1	8080 Model RSP .....	5-2
5.2.2	Small Model RSP .....	5-2
5.3	Multiple Copies of RSPs .....	5-3
5.3.1	8080 Model .....	5-4
5.3.2	Small Model .....	5-4
5.3.3	Small Model with Shared Code .....	5-4
5.4	Creating and Initializing an RSP .....	5-4

## Table of Contents (continued)

5.4.1 The RSP Header .....	5-8
5.4.2 The RSP Process Descriptor .....	5-8
5.4.3 The RSP User Data Area .....	5-9
5.4.4 The RSP Stack .....	5-10
5.4.5 The RSP Command Queue .....	5-10
5.4.6 Multiple Processes within an RSP .....	5-11
5.5 Developing and Debugging an RSP .....	5-12

## 6 System Calls

6.1 System Call Summary .....	6-14
6.2 Concurrent CP/M-86 System Calls .....	6-21
6.2.1 Console I/O System Calls .....	6-22
6.2.2 Device System Calls .....	6-42
6.2.3 Disk Drive System Calls .....	6-45
6.2.4 File-access System Calls .....	6-66
6.2.5 List Device I/O System Calls .....	6-129
6.2.6 Memory System Calls .....	6-135
6.2.7 Process/Program System Calls .....	6-146
6.2.8 Queue System Calls .....	6-171
6.2.9 System Information System Calls .....	6-182

## Appendixes

A Glossary .....	A-1
B ASCII and Hexadecimal Conversions .....	B-1
C Error Codes .....	C-1
D ECHO.A86 Listing .....	D-1

# Table of Contents (continued)

## Tables

1-1.	Registers Used by System Calls .....	1-15
2-1.	File System Calls .....	2-4
2-2.	Valid Filename Delimiters .....	2-6
2-3.	Filetype Conventions .....	2-8
2-4.	Drive Capacity .....	2-9
2-5.	FCB Field Definitions .....	2-12
2-6.	File Attribute Definitions .....	2-16
2-7.	BDOS Interface Attributes F5' and F6' .....	2-17
2-8.	Directory Label Field Definitions .....	2-20
2-9.	XFCB Field Definitions .....	2-22
2-10.	Password Protection Modes .....	2-23
2-11.	Compatibility Attribute Definitions .....	2-35
2-12.	BDOS Physical Errors .....	2-49
2-13.	BDOS Extended Errors .....	2-50
2-14.	BDOS Error Codes .....	2-52
2-15.	BDOS Physical and Extended Errors .....	2-55
3-1.	Group Descriptors .....	3-3
3-2.	Group Descriptor Fields .....	3-4
4-1.	Concurrent CP/M-86 Memory Models .....	4-1
4-2.	Intel Hex Field Definitions .....	4-13
6-1.	System Call Categories .....	6-2
6-2.	Concurrent CP/M-86 System Calls .....	6-4
6-3.	System Call Summary .....	6-14
6-4.	Data Structures Index .....	6-18
6-5.	CX Error Code Reports .....	6-20
6-6.	ACB Field Definitions .....	6-23
6-7.	C_RAWIO Calling Values .....	6-32
6-8.	Console Buffer Field Definitions .....	6-35
6-9.	C_READSTR Line-editing Characters .....	6-35
6-10.	DPB Field Definitions .....	6-50
6-11.	PFCB Field Definitions .....	6-92
6-12.	FCB Initialization .....	6-94
6-13.	MCB Field Definitions .....	6-136
6-14.	MPB Field Definitions .....	6-137



# Table of Contents (continued)

## Tables

6-15.	APB Field Definitions .....	6-147
6-16.	Command Line Buffer Field Definitions .....	6-150
6-17.	PD Field Definitions .....	6-155
6-18.	UDA Field Definitions .....	6-160
6-19.	CPB Field Definitions .....	6-168
6-20.	QPB Field Definitions .....	6-171
6-21.	QD Field Definitions .....	6-177
6-22.	SYSDAT Table Data Fields .....	6-189
6-23.	TOD Field Definitions .....	6-195
B-1.	ASCII Symbols .....	B-1
B-2.	ASCII Conversion Table .....	B-2
C-1.	Concurrent CP/M-86 Error Codes .....	C-1

## Figures

1-1.	Concurrent CP/M-86 Virtual/Physical Environments .....	1-1
1-2.	Concurrent CP/M-86 Functional Modules .....	1-3
2-1.	FCB - File Control Block .....	2-11
2-2.	Directory Label Format .....	2-19
2-3.	XFCB - Extended File Control Block .....	2-21
2-4.	Directory Record with SFCB .....	2-25
2-5.	SFCB Subfields .....	2-26
2-6.	Disk System Reset .....	2-45
3-1.	CMD File Header Format .....	3-2
3-2.	Group Descriptor Format .....	3-3
3-3.	Concurrent CP/M-86 Base Page Values .....	3-5
4-1.	Concurrent CP/M-86 8080 Memory Model .....	4-3
4-2.	Concurrent CP/M-86 Small Memory Model .....	4-5
4-3.	Concurrent CP/M-86 Compact Memory Model .....	4-6
4-4.	Intel Hexadecimal File Formats .....	4-12
5-1.	8080 and Small RSP Models .....	5-2
5-2.	RSP Header Format .....	5-3
5-3.	RSP Command Queue Message .....	5-5

# Table of Contents (continued)

## Figures

5-4.	RSP Data Segment .....	5-7
6-1.	ACB - Assign Control Block .....	6-22
6-2.	Console Buffer Format .....	6-34
6-3.	Drive, R/O, or Login Vector Structure .....	6-45
6-4.	DPB - Disk Parameter Block .....	6-50
6-5.	Disk Free Space Field Format .....	6-66
6-6.	PFCB - Parse Filename Control Block .....	6-91
6-7.	MCB - Memory Control Block .....	6-135
6-8.	MPB - Memory Parameter Block .....	6-136
6-9.	MFPB - M_FREE Parameter Block .....	6-139
6-10.	APB - Abort Parameter Block .....	6-146
6-11.	CLI Command Line Buffer .....	6-150
6-12.	PD - Process Descriptor .....	6-154
6-13.	UDA - User Data Area .....	6-159
6-14.	CPB - Call Parameter Block .....	6-167
6-15.	QPB - Queue Parameter Block .....	6-171

# Section 1

## Concurrent CP/M-86 System Overview

### 1.1 Introduction

Concurrent CP/M-86 is a single-user, multitasking operating system that lets you run multiple programs simultaneously by initiating tasks on two or more virtual consoles. It is compatible with the CP/M-86 single-tasking operating system. Applications programs have access to system calls used by Concurrent CP/M-86 to control the multiprogramming environment. As a result, Concurrent CP/M-86 supports extended features, such as communication among and synchronization of independently running processes. Figure 1-1 depicts the relationships between applications programs, virtual environments, virtual consoles, and the physical console.

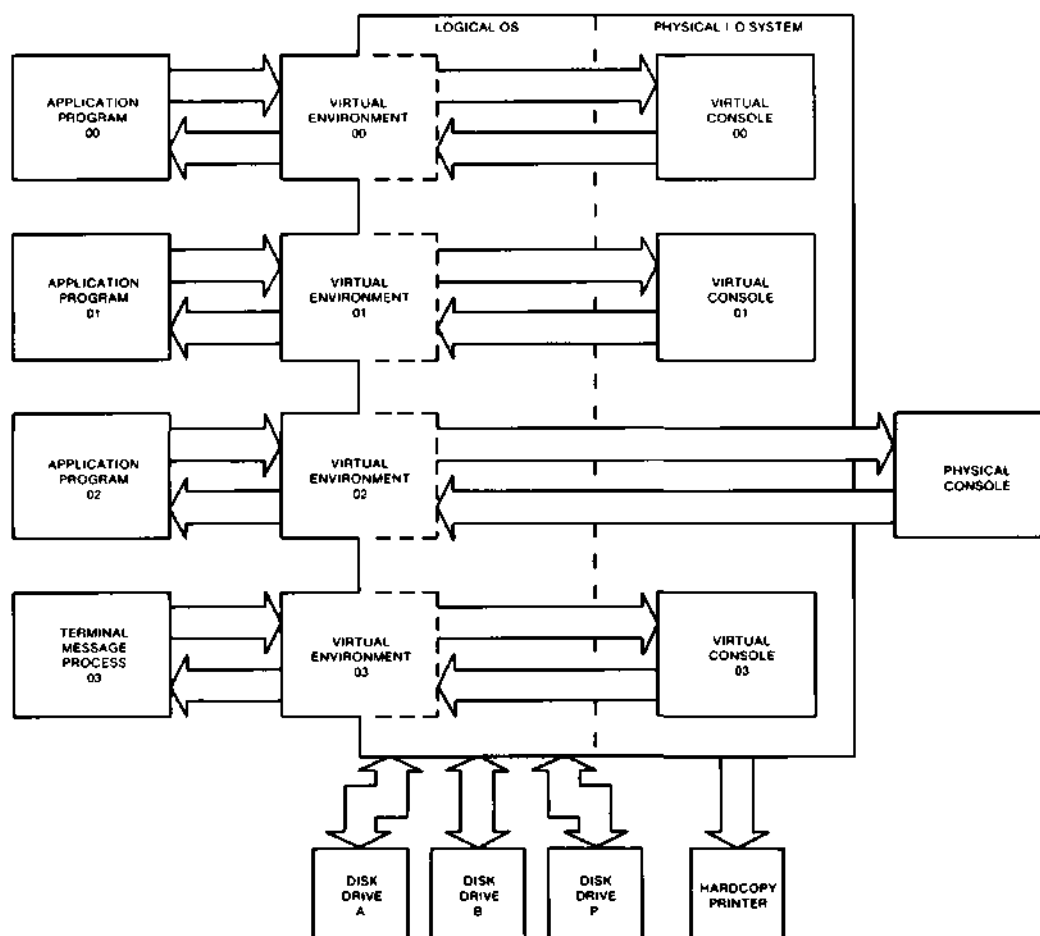


Figure 1-1. Concurrent CP/M-86 Virtual/Physical Environments

In the Concurrent CP/M-86 environment there is an important distinction between a program and a process. A program is simply a block of code residing somewhere in memory or on disk; it is essentially static. A process, on the other hand, is a dynamic entity. You can think of it as a logical machine that executes not only the program code, but also the operating system routines necessary to support the program's functions.

When Concurrent CP/M-86 loads a program, it creates a process associated with the loaded program. Subsequently, it is the process, rather than the program, that obtains access to the system's resources. Thus, Concurrent CP/M-86 monitors the process, not the program. This distinction is a subtle one, but vital to your understanding of system operation as a whole.

Processes running under Concurrent CP/M-86 fall into two categories: transient processes and Resident System Processes (RSPs). Transient processes run programs loaded into memory from disk in response to a user command or system calls made by another process. Resident system processes run code that is a part of the operating system itself. RSPs become an integral part of the operating system image during system generation. They are immediately available to perform operating system tasks. For example, the CLOCK process is an RSP that maintains the time of day within the operating system.

The following list briefly summarizes Concurrent CP/M-86's capabilities.

- Interprocess communication, synchronization, and mutual exclusion functions are provided by system queues.
- A logical interrupt mechanism using flags allows Concurrent CP/M-86 to interface with any physical interrupt structure.
- System timing functions enable processes running under Concurrent CP/M-86 to compute elapsed times, delay execution for specified intervals, and to access and set the current date and time.
- Shared file system allows multiple programs to access common data files while maintaining data integrity.

- Virtual console handling lets a single user run multiple programs, each in its own console environment.
- Real-time process control allows communications and data acquisition without loss of information.

Functionally, Concurrent CP/M-86 is composed of several distinct modules, as shown in Figure 1-2.

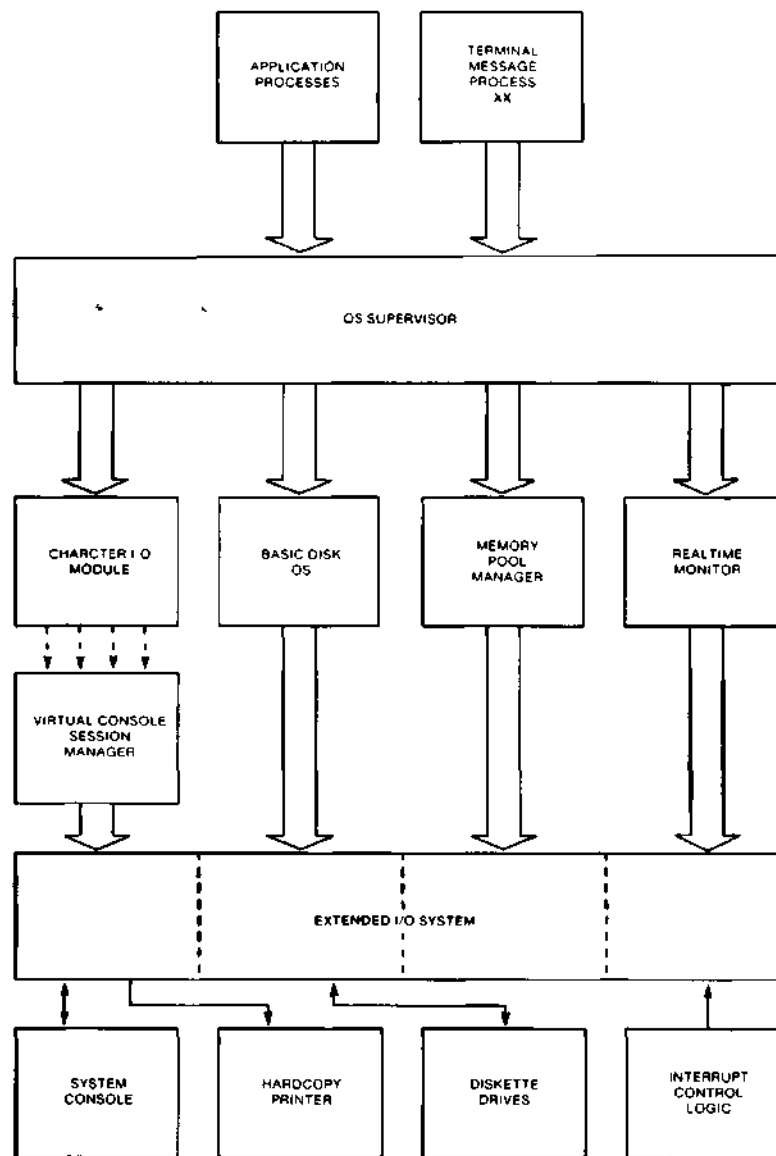


Figure 1-2. Concurrent CP/M-86 Functional Modules

- The Supervisor (SUP)
- The Real-time Monitor (RTM)
- The Memory Management Module (MEM)
- The Character I/O Module (CIO)
- The Virtual Console Screen Manager
- The Basic Disk Operating System (BDOS)
- The Extended I/O System (XIOS)
- The Terminal Message Processor (TMP)

The SUP module handles miscellaneous system calls such as returning the version number or the address of the System Data Area. SUP also calls other system calls when necessary.

The RTM module monitors the execution of running processes and arbitrates conflicts for the system's resources.

The MEM module allocates and frees memory upon demand from executing processes.

The CIO module handles all character I/O for console and list devices in the system.

The Virtual Console Screen Manager extends the CIO to support virtual console environments.

The BDOS is the hardware-independent module that contains the logically invariant portion of the file system for Concurrent CP/M-86. The BDOS file system is explained in detail in Section 2.

The XIOS is the hardware-dependent module that defines the interface of Concurrent CP/M-86 to a specific hardware environment. See the *Concurrent CP/M-86 System Guide* for an explanation of the XIOS.

When Concurrent CP/M-86 is executing a single program on a single virtual console, its speed approximates that of CP/M-86. But when multiple processes are running on several virtual consoles, the execution of each individual process slows according to the proportion of I/O to CPU resources it requires. A process that performs a large amount of I/O in proportion to computing exhibits only minor speed degradation. This also applies to a process that performs a large amount of computing, but runs concurrently with other processes that are largely I/O-bound. On the other hand, significant speed degradation occurs where more than one compute-bound process is running.

## 1.2 Supervisor (SUP)

The Supervisor module (SUP) manages the interface between processes and the operating system kernel. It also manages internal communication between operating system modules. All system calls, whether they originate from a transient process or internally from another system module, go through a common table-driven function interface in SUP. SUP also handles the P\_LOAD (Load Process) and P\_CLI (Call Command Line Interpreter) system calls.

## 1.3 Real-time Monitor (RTM)

The Real-time Monitor (RTM) is the real-time multitasking nucleus of Concurrent CP/M-86. The RTM performs process dispatching, queue management, flag management, device polling, and system timing tasks. User programs can also call many of the RTM system calls used to perform these tasks.

### 1.3.1 Process Dispatching

Although Concurrent CP/M-86 is a multiprocess operating system, only one process has access to the CPU resource at any given time. Unless you specifically write a program to communicate or synchronize execution with other processes, a process is unaware of other processes competing for system resources.

The primary task of the RTM is to transfer, or dispatch, the CPU resource from one process to another. The RTM module called the Dispatcher performs this task. The RTM maintains two data structures, the Process Descriptor (PD) and the User Data Area (UDA), for each process running under Concurrent CP/M-86. The Dispatcher uses these data structures to save and restore the current state of each running process.

Each process in the system resides in one of three states: ready, running, or suspended. A ready process is one that is waiting for the CPU resource only. A running process is one that the CPU is currently executing. A suspended process is one that is waiting for a system resource or a specified event, such as the occurrence of an interrupt, an indication that polled hardware is ready, or the expiration of a delay period.

Any existing process is represented on a system list. The Dispatcher removes a process from one list and places it on another. The Process Descriptor of the currently running process is the first entry on the Ready List. Other processes ready to run are represented on the Ready List in order of priority. Suspended processes are on other System Lists, depending on why the processes were suspended.

A dispatch operation can be summarized as follows:

1. The Dispatcher suspends the process from execution and stores its current state in the Process Descriptor and the UDA.
2. The Dispatcher places the process on an appropriate System List, depending on why the Dispatcher was called. For example, if a process is to delay for a certain number of system ticks, its Process Descriptor is placed on the Delay List. Where a process releases a resource, the process is usually placed back on the Ready List. If another process is waiting for the resource, that process is taken off its current System List and also placed on the Ready List.
3. The highest priority process on the Ready List is chosen for execution. If two or more processes have the same priority, the process that has waited the longest executes first.
4. The Dispatcher restores the state of the selected process from its Process Descriptor and UDA, and gives it the CPU resource.



5. The process executes until it needs a busy resource, a resource needed by another process becomes available, or an interrupt occurs. At this point, a dispatch occurs, allowing another process to run.

Only processes on the Ready List are eligible for selection during dispatch. By definition, a process is on the Ready List if it is waiting only for the CPU resource. Processes waiting for other system resources cannot execute until the resources they require are available. Concurrent CP/M-86 blocks a process from execution if it is waiting for:

- a queue message so it can complete a Q\_READ operation.
- space to become available in a queue so it can complete a Q\_WRITE operation.
- a console or list device to become available.
- a specified number of system clock ticks before it can be removed from the system Delay List.
- an I/O event to complete.

These situations are discussed in greater detail in the following sections.

A running process not needing a resource and not releasing one runs until an interrupt causes a dispatch. While not all interrupts cause dispatches, the system clock generates interrupts every clock tick and forces a dispatch each time. Clock ticks usually occur 60 times a second (approximately every 16.67 milliseconds), and allow time sharing within a real-time environment.

Concurrent CP/M-86 is a priority-driven system. This means that during a dispatch, the operating system gives the CPU resource to the process with the best priority. The Dispatcher allots equal shares of the system's resources to processes with the same priority. With priority dispatching, the system never passes control to a lower-priority process if there is a higher-priority process on the Ready List. Because high-priority, compute-bound processes tend to monopolize the CPU resource, it is best to reduce their priority to avoid degrading overall system performance.

### 1.3.2 Queue Management

Queues perform several critical functions for processes running under Concurrent CP/M-86. A process can use a queue for communicating with another process, synchronizing its execution with that of another process, and for exclusion of other processes from protected system resources. A process can make, open, delete, read from, or write to a queue with system calls similar to those used to manage disk files.

Each system queue consists of two parts: the queue descriptor, and the queue buffer. Concurrent CP/M-86 implements these special data structures as memory files that contain room for a specified number of fixed-length messages.

When the `Q_MAKE` system call creates a queue, this queue is assigned a unique 8-character name. As the name queue implies, messages are read from a queue on a first-in, first-out basis.

A process can read from or write to a queue conditionally or unconditionally. If the queue is empty when a conditional read is performed, or full when a conditional write is performed, the system returns an Error Code to the calling process. On the other hand, if a process attempts an unconditional queue operation in these circumstances, the system suspends it from execution until the operation becomes possible.

More than one process can wait to read or write a queue message from the same queue at the same time. When these operations become possible, the system restores the highest priority process first; processes with the same priority are restored on a first-come, first-served basis.

Mutual exclusion queues are a special type of queue under Concurrent CP/M-86. They contain one message of zero length and their names follow a convention, beginning with the upper-case letters `MX`. A mutual exclusion queue acts as a binary semaphore, ensuring that only one process uses a resource at any time.

Access to a resource protected by a mutual exclusion queue takes place as follows:

1. A process issues an unconditional `Q_READ` call to the `MX` queue protecting the resource, thereby suspending itself if the message is not available.

2. When the message becomes available, the process accesses the protected resource. Note that from the time the process issues the unconditional read, any other process attempting to access the same resource is suspended.
3. The process writes the zero-length message back to the queue when it has finished using the protected resource, thus freeing the resource for other processes.

As an example, the system mutual exclusion queue, MXdisk, ensures that processes cannot access the file system simultaneously. Note that the BDOS, not the application software, executes the preceding series of queue calls. Therefore the mutual exclusion process is transparent to the programmer, who is only responsible for originating the disk system calls.

Mutual exclusion queues differ from normal queues in another way. When a process reads a message from a mutual exclusion queue, the RTM notes the Process Descriptor address within the Queue Descriptor. This establishes the owner of the queue message. If the operating system aborts the process while it owns the mutual exclusion message, the RTM automatically writes the message back to all mutual exclusion queues whose messages are owned by the aborted process. This grants other processes access to protected resources owned by the aborted process.

### 1.3.3 System Timing Functions

Concurrent CP/M-86's timing system calls include keeping the time of day and delaying the execution of a process for a specified period of time. An internal process called CLOCK provides the time of day for the system. This process issues DEV\_WAITFLAG system calls on the system's one second flag, Flag 2. When the XIOS Tick Interrupt Handler sets this flag, it initiates the CLOCK process, which then increments the internal time and date.

Subsequently, the CLOCK process makes another DEV\_WAITFLAG call and suspends itself until the flag is set again. Concurrent CP/M-86 provides system calls that allow you to set and access the internal date and time. In addition, the file system uses the internal time and date to record when a file is updated, created, or last accessed.

The P\_DELAY system call replaces the typical programmed delay loop for delaying process execution. P\_DELAY requires that Flag 1, the system tick flag, be set approximately every 16.67 milliseconds, or 60 times a second; the XIOS Tick Interrupt Handler also sets this flag. When a process makes a P\_DELAY system call, it specifies the number of ticks for which the operating system is to suspend it from execution. The system maintains the address of the Process Descriptor for the process on an internal Delay List along with its current delay tick count. When a DEV\_SETFLAG call occurs, setting flag 1, the tick count is decremented. When the delay count goes to zero, the system removes the process from the Delay List and places it on the Ready List.

**Note:** the length of a tick might vary from installation to installation. For instance, in Europe, a tick is commonly 20 milliseconds, yielding 50 ticks per second. The description of the P\_DELAY system call in Section 6 describes how to determine the correct number of ticks to delay 1 second.

## 1.4 Memory Module (MEM)

Concurrent CP/M-86 supports an extended, fixed partition model of memory management; the Memory Module handles all memory management system calls. In practice, the exact method that the operating system uses to allocate and free memory is transparent to the application program. Therefore you should take care to write code independent of the memory management model; use only the Concurrent CP/M-86-specific memory system calls described in Section 6.

## 1.5 Basic Disk Operating System (BDOS)

a.1

The Concurrent CP/M-86 BDOS is an upward-compatible version of the single-tasking CP/M-86 BDOS. It handles file creation and deletion, facilitates sequential or random file access, and allocates and frees disk space. In most cases, CP/M-86 programs that make BDOS calls for I/O can run under Concurrent CP/M-86 without modification. Concurrent CP/M-86's BDOS is extended to provide support for multiple virtual consoles and list devices. In addition, the file system is extended to provide services required in a multitasking environment. The major extensions to the file system are

- **File locking.** Files opened under Concurrent CP/M-86 cannot be opened or deleted by other tasks. This feature prevents accidental conflicts with other tasks.
- **Shared access to files.** As a special option, independent users can open the same file in shared or unlocked mode. Concurrent CP/M-86 supports record locking and unlocking commands for files opened in this mode and protects files opened in shared mode from deletion by other tasks.
- **Date Stamps.** The BDOS optionally supports two time and date stamps, one recording when a file is updated, and the other recording when the file was created or last accessed.
- **Password Protection.** The password protection feature is optional at either the file or drive level. The operator or applications program assigns disk drive passwords, while application programs can assign file protection passwords in several modes.
- **Extended Error Module.** Besides the default error mode, Concurrent CP/M-86 has two optional error-handling modes that return an error code to the calling process in the event of an irrecoverable disk error.

## 1.6 Character I/O Module (CIO)

The Character I/O module handles all console and list I/O. Under Concurrent CP/M-86, every character I/O device is associated with a data structure called a Console Control Block (CCB) or a List Control Block (LCB). These data structures reside in the XIOS. The CCB contains the current owner, status information, line editing variables, and the root of a linked list of Process Descriptors (PDs) that are waiting for access. More than one process can wait for access to a single console. These processes are maintained on a linked list of Process Descriptors in priority order. The LCBs contain similar information about the list devices. See the *Concurrent CP/M-86 System Guide* for more information about LCBs and CCBs.

## 1.7 Virtual Console Screen Management

Virtual console screen management is coordinated by four separate modules: the CIO, the PIN (Physical INput) and VOUT (Virtual OUTput) processes, and the XIOS. The line editing associated with the C\_READSTR call is performed in the CIO. The PIN process handles keyboard input for all the virtual consoles; it also traps and implements the CTRL-C, CTRL-S, CTRL-Q, CTRL-P, and CTRL-O functions. The VOUT process spools console output from processes running on background buffered mode consoles, and handshakes with the PIN process to display spooled console output when the background console is brought to the foreground. The XIOS decides which special keys represent the virtual consoles, and returns a special code from IO\_CONIN when you request a screen switch. The XIOS also implements any screen saving and restoring when screens are switched. See the *Concurrent CP/M-86 System Guide* and the discussion of the IO\_SWITCH function.

The PIN process reads the keyboard by directly calling the XIOS IO\_CONIN function. This is the only place in the operating system IO\_CONIN is called. The PIN scans the input stream from the keyboard for switch screen requests and the special function keystrokes CTRL-C, CTRL-S, CTRL-Q, CTRL-P, and CTRL-O. All other keyboard input is written to the VINC (Virtual Console INput Queue) associated with the foreground virtual console. The data in the VINC becomes a type-ahead buffer for each virtual console, and is returned to the process attached to that console as it performs console input.

When PIN sees a CTRL-C it calls P\_ABORT to abort the process attached to the virtual console, flushes the type-ahead buffer in the VINC, turns off CTRL-S, and performs a DRV\_RESET call for each logged-in drive. The P\_ABORT call succeeds when the Process Keep flag is not on, saving the Terminal Message Processes (refer to P\_CREATE for information on the process descriptor). The DRV\_RESET calls affect only the removable media drives, as specified in the CKS field of the Disk Parameter Blocks in the XIOS (refer to the *Concurrent CP/M-86 System Guide* for further details on Disk Parameter Blocks).

CTRL-S stops any output to the screen. CTRL-S stays set when a virtual console is switched to the background.

CTRL-O discards any console output to the virtual console. CTRL-O is turned off when any other key is subsequently pressed, except for the keys representing the virtual consoles.

CTRL-P echoes console output to the default list device specified in the LIST field of the process descriptor attached to the virtual console. If the list device is attached to a process, a PRINTER BUSY message appears.

All of the above control keys can be disabled by the C\_MODE call. When one of the above control characters is disabled with C\_MODE or when the process owning the virtual console is using the C\_RAWIO call, the PIN does not act on the control character but instead writes it to the VINC. It is thus possible to read any of the above control characters from an application program. These control keys are discussed in depth in the *Concurrent CP/M-86 User's Guide*.

## 1.8 Extended Input/Output System (XIOS)

The XIOS module is similar to the CP/M-86 Basic Input/Output System (BIOS) module, but it is extended in several ways. Primitive operations, such as console I/O, are modified to support multiple virtual consoles. Several new primitive system calls, such as DEV\_POLL, support Concurrent CP/M-86's additional features, including elimination of wait loops for real-time I/O operations.

## 1.9 Terminal Message Processes (TMP)

The Concurrent CP/M-86 Terminal Message Processes (TMPs) are resident system processes that accept command lines from the virtual consoles and call the Command Line Interpreter (CLI) to execute them. The TMP prints the prompt on the virtual consoles.

Each virtual console has an independent TMP defining that console's environment, including default disk, user number, printer, and console.

## 1.10 Transient Programs

Under Concurrent CP/M-86, a transient program is one that is not system-resident. The system must load such programs from disk into available memory every time it executes. The command file of a transient program is identified by the filetype CMD. When you enter a command at the console, the operating system searches on disk for the appropriate CMD file, loads it, and initiates it. Concurrent CP/M-86 supports three different execution models for transient programs: the 8080 Model, the Small Model, and the Compact Model. Sections 4.1.1 through 4.1.3 describe these models in detail.



## 1.11 System Call Calling Conventions

When a Concurrent CP/M-86 process makes a system call, it loads values into the registers shown in Table 1-1 and initiates Interrupt 224 (via the INT 224 instruction), reserved by the Intel® Corporation for this purpose.

**Table 1-1. Registers Used by System Calls**

ENTRY PARAMETERS	
Register	CL: System Call Number
	DL: Byte Parameter
	or
	DX: Word Parameter
	or
	DX: Address - Offset
	DS: Address - Segment
RETURN VALUES	
Register	AL: Byte Return
	or
	AX: Word Return
	or
	AX: Address - Offset
	ES: Address - Segment
	BX: Same as AX
	CX: Error Code

Concurrent CP/M-86 preserves the contents of registers SI, DI, BP, SP, SS, DS, and CS through the operating system calls. The ES register is preserved when it is not used to hold a return segment value. Error codes returned in CX are shown in Table 6-5, CX Error Codes.

## 1.12 SYSTAT: System Status

The SYSTAT utility is a development tool that shows the internal state of Concurrent CP/M-86. SYSTAT describes memory allocation, current processes, system queue activity, and many informative parameters associated with these system data structures. Furthermore, SYSTAT presents two views: either a static snapshot of system activity, or a continuous, real-time window into Concurrent CP/M-86.

You can specify SYSTAT in one of two modes. If you know which display you want, you can specify it in the invocation, using an option shown in the menu below. If you do not specify an option, select a display from this menu by typing

```
A>SYSTAT <cr>
```

The screen clears and the main menu appears:

Which Option?

```
H(elp)
M(emory)
O(overview)
P(rocesses - All)
Q(ueues)
U(ser Processes)
E(xit)
```

```
->_
```

Press the appropriate letter to obtain a display.

When you select H(elp), the HELP file demonstrates the proper syntax and available options:

To use SYSTAT with the menu: At the system prompt type SYSTAT <CR>

To use SYSTAT without the menu: At the system prompt type the command

```
SYSTAT [option] -or-  
SYSTAT [option C] -or-  
SYSTAT [option C **]
```

-where-

```
-> option =  
    M(emory) P(rocesses) O(overview)  
    U(scr Processes) Q(ueues) H(elp)  
  
-> C = Continuous display  
    ** = 1-2 digits indicating the period,  
        in seconds, between display refreshes.
```

Type any letter to return to the menu.

The M, P, Q, and U options ask you if you prefer a continuous display. If you type y, Concurrent CP/M-86 asks for a time interval, in seconds, and then displays a real-time window of information. If you type n, a static snapshot of the requested information appears. In either case, press any key to return to the menu.

The M(emory) option displays all memory potentially available to you, but it does not display restricted memory. The partitions are listed in memory-address order. Length parameter is shown in paragraph values.

The P(rocess) option displays all system processes and the resources they are using.

The Q(ueues) option displays all system queues, listing queue readers, writers, and owners.

The U(ser Processes) option displays only user-initiated processes in the same format as the P(rocess) option.

The O(overview) option displays an overview of the system parameters, as specified at system generation time. The display is not continuous.

The E(xit) option returns you to system level from the menu, as does CTRL-C.

*End of Section 1*

## Section 2

# The Concurrent CP/M-86 File System

### 2.1 File System Overview

The Basic Disk Operating System (BDOS) file system supports from one to sixteen logical drives. Each logical drive 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 directory area consists of sixteen logically independent directories. These directories are identified by user numbers 0 through 15. During execution, a process runs with a system parameter called the user number set to a single value. The user number specifies the current active directories for all drives on the system. For example, the Concurrent CP/M-86 DIR utility displays only files within a directory selected by the current user number.

The file system automatically allocates directory and data area space when a process creates or extends a file, and returns previously allocated space to free space when a process deletes or truncates a file. If no directory or data space is available for a requested operation, the BDOS returns an error code to the calling process. The allocation and retrieval of directory and data space is transparent to the calling process. As a result, you need not be concerned with directory and drive organization when using the file system calls.

An eight-character filename and a three-character filetype field identify each file in a directory. Together, these fields must be unique for each file within a directory. However, files with the same filename and filetype can reside in different user directories without conflict. Processes 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. For some system calls, multiple files can be specified by a technique called ambiguous reference. This technique uses question marks and asterisks as wildcard characters to give the file system a pattern to match as it searches a directory.

The file system supports two categories of system calls: file-access system calls and drive-related system calls. The file-access system calls have mnemonics beginning with `F_`, and the drive-related system calls have mnemonics beginning with `DRV_`. The next two sections introduce the file system calls.

### 2.1.1 File-access System Calls

Most of the file-access system calls can be divided into two groups: system calls that operate on files within a directory and system calls that operate on records within a file. However, the file-access category also includes several miscellaneous functions that either affect the execution of other file-access system calls or are commonly used with them.

System calls in the first file-access group include calls to search for one or more files, delete 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. Also included in this group are system calls to open a file, to create a file, and to close a file.

The second file-access group includes system calls to read or write records to a file, either sequentially or randomly, by record position. BDOS read and write system calls transfer data in 128 byte units, which is the basic record size of the file system. This group also includes system calls to lock and unlock records and thereby allow multiple processes to coordinate access to records within a commonly accessed file.

Before making read, write, lock, or unlock system calls for a file, you must first open or create the file. Creating a file has the side effect of opening the file for record access. In addition, because Concurrent CP/M-86 supports three different modes of opening files (Locked, Unlocked, and Read-Only), there can be other restrictions on system calls in this group that are related to the open mode. For example, you cannot write to a file that you have opened in Read-Only mode.

After a process has opened a file, access to the file by other processes is restricted until the file is closed. Again, the exact nature of the restrictions depends on the open mode. However, in all cases the file system does not allow a process to delete, rename, or change a file's attributes if another process has opened the file. Thus, the close system call performs two steps to terminate record access to a file. It permanently records the current status of the file in the directory and removes the open-file restrictions limiting access to the file by other processes.

The miscellaneous file-access system calls include calls to set the current user number, set the DMA address, parse an ASCII file specification and set a default password. This group also includes system calls to set the BDOS Multisector Count and the BDOS Error Mode. The BDOS Multisector count determines the number of 128-byte records to be processed by the read, write, lock, and unlock system calls. The Multisector count can range from 1 to 128; the default value is one. The BDOS Error Mode determines whether the file system intercepts certain errors or returns on all errors to the calling process.

### 2.1.2 Drive-related System Calls

BDOS drive-related system calls select the default drive, compute a drive's free space, interrogate drive status, and assign a directory label to a drive. A drive's directory label controls whether the file system enforces file password protection for files in the directory. It also specifies whether the file system is to perform date and time stamping of files on the drive.

This category also includes system calls to reset specified drives and to control whether other processes can reset particular drives. When a drive is reset, the next operation on the drive reactivates it by logging it in. Logging in a drive initializes the drive for directory and file operations. The purpose of a drive reset call is to prepare for a media change on drives that support removable media. Under Concurrent CP/M-86, drive reset calls are conditional. A process cannot reset a drive if another process has a file open on the drive.

The following table summarizes the BDOS file system calls.

Table 2-1. File System Calls

<i>Mnemonic</i>	<i>Description</i>
DRV_ACCESS	Access Drive
DRV_ALLOCVEC	Get Drive Allocation Vector
DRV_ALLRESET	Reset All Drives
DRV_DPB	Get Disk Parameter Block Address
DRV_GET	Get Default Drive
DRV_GETLABEL	Get Directory Label
DRV_FLUSH	Flush Data Buffers
DRV_FREE	Free Drive
DRV_LOGINVEC	Return Drives Logged In Vector
DRV_RESET	Reset Drive
DRV_ROVEC	Return Drives R/O Vector
DRV_SETLABEL	Set Directory Label
DRV_SET	Set (Select) Drive
DRV_SETRO	Set Drive To Read-Only
DRV_SPACE	Get Free Space On Drive
F_ATTRIB	Set File's Attributes
F_CLOSE	Close File
F_DELETE	Delete File
F_DMASEG	Set DMA Segment
F_DMAGET	Get DMA Address
F_DMAOFF	Set DMA Offset
F_ERRMODE	Set BDOS Error Mode
F_LOCK	Lock Record In File
F_MAKE	Make A New File
F_MULTISEC	Set BDOS Multisector Count
F_OPEN	Open File
F_PARSE	Parse Filename
F_PASSWD	Set Default Password



Table 2-1. (continued)

<i>Mnemonic</i>	<i>Description</i>
F_RANDOMREC	Return Record Number For File Read-Write
F_READ	Read Record Sequentially From File
F_READRAND	Read Random Record From File
F_RENAME	Rename File
F_SIZE	Compute File Size
F_SFIRST	Directory Search First
F_SNEXT	Directory Search Next
F_TIMEDATE	Return File Time/Date Stamps Password Mode
F_TRUNCATE	Truncate File
F_UNLOCK	Unlock Record In File
F_USERNUM	Set/Get Directory User Number
F_WRITE	Write Record Sequentially Into File
F_WRITERAND	Write Random Record Into File
F_WRITEZF	Write Random Record With Zero Fill
F_WRITEXFCB	Write File's XFCB

The following sections contain information on important topics related to the file system. Read these sections carefully before attempting to use the system calls described individually in Section 6.

## 2.2 File Naming Conventions

Under Concurrent CP/M-86, a file specification consists of four parts: a drive specifier, the filename field, the filetype field, and the file password field. The general format for a command line file specification is shown below:

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

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

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

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

Under Concurrent CP/M-86, the P\_CLI system call interprets ASCII command lines and loads programs. The P\_CLI system call makes F\_PARSE system calls to parse file specifications from a command line. F\_PARSE recognizes certain ASCII characters as delimiters when it parses a file specification. These characters are shown in Table 2-2.

**Table 2-2. Valid Filename Delimiters**

<i>ASCII</i>	<i>Hex Equivalent</i>
null	000H
space	020H
return	00DH
tab	009H
:	03AH
.	02EH
;	03BH
=	03DH
,	02CH
[	05BH
]	05DH
<	03CH
>	03EH
	07CH

The `F_PARSE` system call also excludes all control characters from the file specification fields and translates all lower-case letters to upper-case.

Avoid using parentheses and the backslash character, `\`, in the filename and filetype fields because they are commonly used delimiters. Use asterisk and question mark characters, `*` and `?`, only to make an ambiguous file reference. When `F_PARSE` encounters an asterisk in a filename or filetype field, it pads the remainder of the field with question marks. For example, a filename of `X*.*` is parsed to `X???????.*`. The BDOS `F_SFIRST`, `F_SNEXT`, and `F_DELETE` system calls match a question mark in the filename or filetype fields to the corresponding position of any directory entry belonging to the current user number. Thus, a search operation for `X???????.*` finds all the files in the current user directory beginning in `X`. Most other file-access BDOS system calls treat the presence of a question mark in the filename or filetype fields as an error.

It is not mandatory to follow the file naming conventions of Concurrent CP/M-86 when you create or rename a file with BDOS system calls directly from an application program. However, the conventions must be used if the file is to be accessed from a command line. For example, the `P_CLI` system call cannot locate a command file in the directory if its filename or filetype field contains a lower-case letter.

As a general rule, the filetype field names the generic category of a particular file, and the filename field distinguishes individual files within each category. Although they are generally arbitrary, Table 2-3 lists some of the generic filetype categories that have been established.

Table 2-3. Filetype Conventions

<i>Filetype</i>	<i>Description</i>
A86	8086 Assembler Source
ASM	Assembler Source
BAK	Text or Source Back-up
BAS	BASIC Source File
C	C Source File
CMD	8086 Command File
COM	8080 Command File
CON	CCP/M-86 Modules
DAT	Data File
HEX	HEX Machine Code
H86	ASM86 HEX File
INT	Intermediate File
LIB	Library File
LST	List File
PLI	PL/I Source File
PRL	Page Relocatable
REL	Relocatable Module
RSP	Resident System Process
SPR	System Page Relocatable
SUB	SUBMIT File
SUP	Startup File
SYM	Symbol File
SYS	System File
\$\$\$	Temporary File

## 2.3 Disk Drive and File Organization

The file system can support up to sixteen logical drives, identified by the letters A through P. 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. If a disk contains system tracks reserved for the boot loader, these tracks precede the tracks of the disk mapped by the logical drive. In this manual, references to drives means logical drives, unless explicitly stated otherwise.

The maximum file size supported on a drive is 32 megabytes. The maximum capacity of a drive is determined by the data block size specified for the drive in the XIOS. The data block size is the basic unit in which the BDOS allocates space to files. Table 2-4 displays the relationship between data block size and total drive capacity.

Table 2-4. Drive Capacity

<i>Data Block Size</i>	<i>Maximum Drive Capacity</i>
1K	256 kilobytes
2K	64 megabytes
4K	128 megabytes
8K	256 megabytes
16K	512 megabytes

Each drive is divided into two regions: a directory area and a data area. The directory area contains from one to sixteen blocks located at the beginning of the drive. The actual number is set in the XIOS. Directory entries residing in this area define the files that exist on the drive. In addition, the directory entries belonging to a file identify the data blocks in the drive's data area that contain the file's records. The directory area is logically subdivided into sixteen independent directories identified as user 0 through 15. Each independent directory shares the actual directory area on the drive.

Each disk file consists of a set of up to 262,144 (40000H) 128-byte records. Each record of a file is identified by its position in the file. This position is called the record's Random Record Number. If a file is created sequentially, the first record has a position of zero, while the last record has a position one less than the number of records in the file. Such a file can be read sequentially, beginning at record zero, or randomly by record position. Conversely, if a file is created randomly, records are added to the file by specified position. A file created in this way is called sparse if positions exist within the file where a record has not been written.

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

Source files under Concurrent CP/M-86 are treated as a sequence of ASCII characters, where each line of the source file is followed by a carriage return/line-feed sequence, 0DH followed by 0AH. Thus, a single 128-byte record could contain several lines of source text. The end of an ASCII file is denoted by a CTRL-Z character (1AH), or a real end-of-file, returned by the BDOS read system call. Note that these source file conventions are not supported in the file system directly but are followed by Concurrent CP/M-86 utilities such as TYPE and ASM-86™. In addition, CTRL-Z characters embedded within other types of files such as CMD files do not signal end-of-file.

## 2.4 File Control Block Definition

The File Control Block (FCB) is a system data structure that serves as an important channel for information exchange between a process and BDOS file-access system calls. A process initializes an FCB to specify the drive location, filename and filetype fields, and other information that is required to make a file-access call. For example, in an F\_OPEN system call, the FCB specifies the name and location of the 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 BDOS random I/O system calls require the calling process to specify the Random Record Number in a 3-byte field at the end of the FCB.

When a process makes a BDOS file-access system call, it passes an FCB address to the BDOS. This address has two 16-bit components: register DX, which contains the offset, and register DS, which contains the segment. The length of the FCB data area depends on the BDOS system call. For most system calls, the minimum length is 33 bytes. For the F\_READRAND, F\_WRITERAND, F\_WRITEZF, F\_LOCK, F\_UNLOCK, F\_RANDREC, F\_SIZE, and F\_TRUNCATE system calls, the minimum FCB length is 36 bytes. When the F\_OPEN or F\_MAKE system calls open a file in Unlocked mode, the FCB must be at least 35 bytes long. Figure 2-1 displays the FCB data structure in two formats.

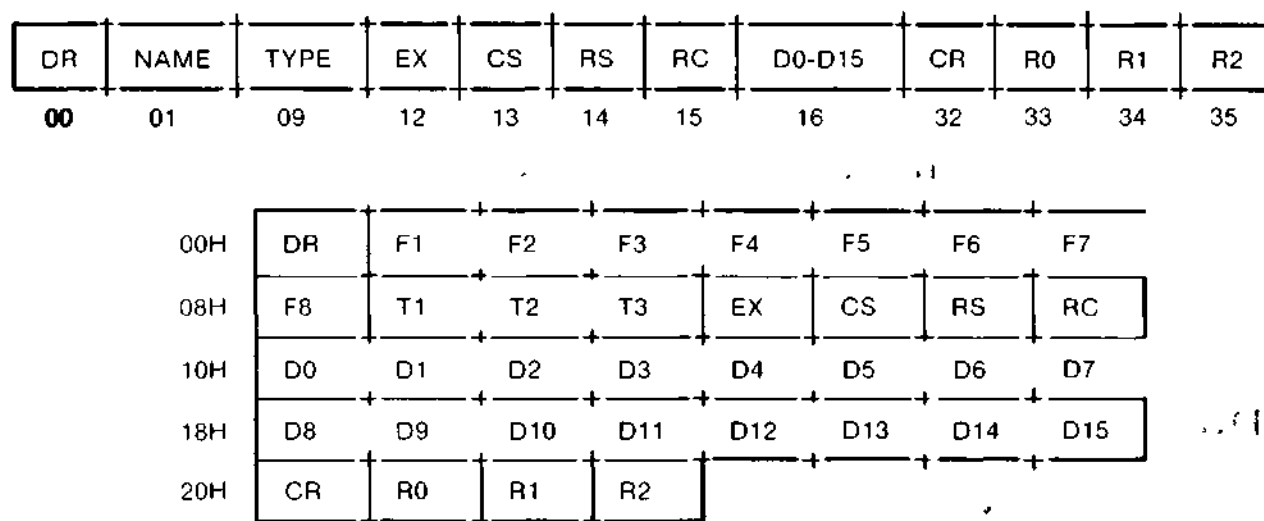


Figure 2-1. FCB - File Control Block

The fields in the FCB are defined as follows:

**Table 2-5. FCB Field Definitions**

<i>Field</i>	<i>Definitions</i>
DR	<p>Drive Code (0 - 16).</p> <p>0 =&gt; use default drive for file  1 =&gt; auto disk select drive A  2 =&gt; auto disk select drive B  . . .  . . .  . . .  16 =&gt; auto disk select drive P</p>
F1...F8	<p>Contain the filename in ASCII upper-case, with high bit = 0. F1', ..., F8' denote the high-order bit of these positions and are called attribute bits.</p>
T1,T2,T3	<p>Contain the filetype in ASCII upper-case, with high bit = 0. T1', T2', and T3' denote the high bit of these positions and are also called attribute bits.</p> <p>T1' = 1 =&gt; Read-Only file,  T2' = 1 =&gt; System file,  T3' = 1 =&gt; File has been archived.</p>
EX	<p>Contains the current extent number. This field is usually set to 0 by the calling process, but it can range from 0 to 31 during file I/O.</p>
CS	<p>Contains the FCB checksum value for open FCBs.</p>



Table 2-5. (continued)

<i>Field</i>	<i>Definitions</i>
RS	Reserved for internal system use.
RC	Record count for extent EX. This field takes on values from 0 to 255 (values greater than 128 imply a record count of 128).
D0...D15	Normally filled in by Concurrent CP/M-86 and reserved for system use. Also used to specify the new filename and filetype with the F_RENAME system call.
CR	Current record to read or write in a sequential file operation. This field is normally set to zero by the calling process when a file is opened or created.
R0,R1,R2	Optional Random Record Number in the range 0-262,143 (0 - 3FFFFH). R0, R1, R2 constitute an 18-bit value with low byte R0, middle byte R1, and high byte R2.

**Note:** the 2-byte File ID is returned in bytes R0 and R1 of the FCB when a file is successfully opened in Unlocked mode (refer to Section 2.14).

### 2.4.1 FCB Initialization and Usage

The calling process must initialize bytes 0 through 11 of the referenced FCB before making the following file-access system calls: F\_ATTRIB, F\_DELETE, F\_MAKE, F\_OPEN, F\_RENAME, F\_SFIRST, F\_SIZE, F\_SNEXT, F\_TIMEDATE, F\_TRUNCATE, and F\_WRITEFCB. Normally, the DR field specifies the drive location of the file, and the name and type fields specify the name of the file. You must also set the EX field of the FCB before calling F\_MAKE, F\_OPEN, F\_SFIRST, and F\_WRITEFCB. Except for the F\_WRITEFCB system call, you can usually set this field to zero. Note that the F\_RENAME system call requires the calling process to 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` system call expects an FCB initialized by a prior `F_SFIRST` call. In addition, the `F_LOCK`, `F_READ`, `F_READRAND`, `F_UNLOCK`, `F_WRITERAND`, and `F_WRITEZF` system calls require an FCB that has been activated for record operations. Under Concurrent CP/M-86, only the `F_OPEN` and `F_MAKE` system calls can activate an FCB.

If you intend to process a file sequentially from the beginning, using the `F_READ` and `F_WRITE` system calls, you must set byte 32 to zero before you make your first read or write call. In addition, when you make a `F_LOCK`, `F_READRAND`, `F_UNLOCK`, `F_WRITERAND`, or `F_WRITEZF` system call, you must set bytes R0 through R2 of the FCB to the requested Random Record Number. The `F_TRUNCATE` system calls also requires the FCB random record field to be initialized.

The `F_SFIRST`, `F_SNEXT`, and `F_DELETE` system calls support multiple or ambiguous reference. In general, a question mark in the filename, filetype, or EX fields matches all values in the corresponding positions of directory entries during a directory search operation. File directory entries 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 not present. The search system calls, `F_SFIRST` and `F_SNEXT`, also recognize a question mark in the FCB DR field, and, if specified, they return all directory entries on the disk regardless of user number, including empty entries. A directory FCB that begins with E5H is an empty directory entry.

When the `F_OPEN` and `F_MAKE` system calls activate an FCB for record operations, they copy the FCB's matching directory entry from disk, excluding byte 0, into the FCB in memory. In addition, these system calls compute and store a checksum value in the CS field of the FCB. During subsequent record operations on the file, the file system uses this checksum field to verify that the FCB has not been modified by the calling process in an illegal way. Thus, all read, write, lock, and unlock operations on a file must specify a valid activated FCB; otherwise, the BDOS returns a checksum error. The BDOS performs this checking to protect the integrity of the file system. In general, you should not modify bytes 0 through 31 of an open FCB, except to set interface attributes (see Section 2.4.3). Other restrictions related to activated FCBs are discussed in Section 2.10.

The BDOS updates the memory copy of the FCB during file processing to maintain the current position within the file. During file write operations, the BDOS also updates the memory copy of the FCB to record the allocation of data blocks to the file. At the termination of file processing, the F\_CLOSE system call permanently records this information on disk.

Note that the BDOS does not record the data blocks allocated to a file during write operations in the disk directory until the calling process issues an F\_CLOSE call. Therefore, a process that creates or modifies files must close the files at the termination of file processing. Otherwise, data might be lost.

### 2.4.2 File Attributes

The high-order bits of the FCB filename (F1',...,F8') and filetype fields (T1',T2',T3') are called attribute bits. Attribute bits are 1-bit Boolean fields, where 1 indicates on or true, and 0 indicates off or false. Attribute bits indicate two kinds of attributes within the file system: file attributes and interface attributes. The file attributes are described in this section. Section 2.4.3 describes interface attributes.

The file attribute bits, F1',...,F4' and T1', T2', T3', indicate that a file has a defined attribute. These bits are recorded in a file's directory FCBs. File attributes can be set or reset only by the F\_ATTRIB system call. When the F\_MAKE system call creates a file, it initializes all file attributes to zero. A process can interrogate file attributes in an FCB activated by the F\_OPEN system call, or in directory FCBs returned by the F\_SFIRST and F\_SNEXT system calls.

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

The file system defines file attributes T1', T2', and T3' as follows:

Table 2-6. File Attribute Definitions

<i>Attribute</i>	<i>Definition</i>
T1': Read-Only Attribute	This attribute, if set, prevents write operations to a file.
T2': System Attribute	This attribute, if set, identifies the file as a Concurrent CP/M-86 system file. The Concurrent CP/M-86 DIR utility does not usually display System files. In addition, user-zero system files can be accessed on a Read-Only basis from other user numbers.
T3': Archive Attribute	User-written archive programs use this attribute. When an archive program copies a file to back-up storage, it sets the archive attribute of the copied files. The file system automatically resets the archive attribute of a directory entry when writing to the directory entry's region of a file. An archive program can test this attribute in each of the file's directory entries using the F_SFIRST and F_SNEXT system calls. If all directory entries have the archive attribute set, the file has not been modified since the previous archive. The Concurrent CP/M-86 PIP utility supports file archival.

File attributes F1' through F4' of command files are defined as Compatibility Attributes under Concurrent CP/M-86 (see Section 2.12). However, for all other files, attributes F1' through F4' are available for definition by the user.

### 2.4.3 Interface Attributes

The interface attributes are F5', F6', F7', and F8'. These attributes cannot be used as file attributes. Interface attributes F5' and F6' request options for BDOS file-access system calls. Table 2-7 lists the F5' and F6' attribute definitions for the system calls that define interface attributes. Note that the F5' = 0 and F6' = 0 definitions are not listed if their definition simply implies the absence of the option associated with setting the interface attribute.

**Table 2-7. BDOS Interface Attributes F5' and F6'**

<i>System Call</i>	<i>Attribute</i>
F_ATTRIB	F5' = 1 : Maintain extended file lock F6' = 1 : Set file byte count
F_CLOSE	F5' = 1 : Partial Close F6' = 1 : Extend file lock
F_DELETE	F5' = 1 : Delete file XFCBs only and maintain extended file lock
F_LOCK	F5' = 0 : Exclusive Lock F5' = 1 : Shared Lock F6' = 0 : Lock existing records only F6' = 1 : Lock logical records
F_MAKE	F5' = 0 : Open in Locked mode F5' = 1 : Open in Unlocked mode F6' = 1 : Assign password to file
F_OPEN	F5' = 0 : Open in Locked mode F5' = 1 : Open in Unlocked mode F6' = 0 : Open in mode specified by F5' F6' = 1 : Open in Read-Only mode
F_RENAME	F5' = 1 : Maintain extended file lock
F_TRUNCATE	F5' = 1 : Maintain extended file lock
F_UNLOCK	F5' = 1 : Unlock all locked records

Section 6 details the above interface attribute definitions for each of the preceding system calls. Note that the BDOS always resets interface attributes F5' and F6' before returning to the calling process. Interface attributes F7' and F8' are reserved for internal use by the file system.

## 2.5 User Number Conventions

The Concurrent CP/M-86 user facility divides each drive directory into sixteen logically independent directories, designated as user 0 through user 15. Physically, all user directories share the directory area of a drive. In most other aspects, however, they are independent. For example, files with the same name can exist on different user numbers of the same drive with no conflict. However, a single file cannot extend across more than one user number.

Only one user number is active for a specific process at one time. For this process, the current user number applies to all drives on the system. Furthermore, the FCB format does not contain a field that can override the current user number. As a result, all file and directory operations reference only directory entries associated with the current user number.

It is possible for a process to access files on different user numbers by setting the user number to the file's user number with the F\_USERNUM system call before issuing the BDOS call. However, if a process attempts to read or write to a file under a user number different from the user number that was active when the file was opened, the file system returns an FCB checksum error.

When the P\_CLI system call initiates a transient process or Resident System Process (described in detail in Section 5), it sets the user number to the default value established by the process issuing the P\_CLI system call. The sending process is usually the TMP. However, the sending process can be another process, such as a transient program that makes a P\_CHAIN call. A transient process can change its user number by making a F\_USERNUM call. Changing the user number in this way does not affect the command line user number displayed by the TMP. Thus, when a transient process that has changed its user number terminates, the TMP restores and displays the original user number in the command line prompt when it regains control.

User 0 has special properties under Concurrent CP/M-86. The file system automatically opens files listed under user zero but requested under another user number if the file is not present under the current user number, and if the file on user zero has the system attribute (T2') set. 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, and gives the Concurrent CP/M-86 user control over which files are accessible to the different user areas.

## 2.6 Directory Labels and XFCBs

The file system includes three special types of FCBs: the directory label and the XFCB, described in this section, and the SFCB, described in detail in Section 2.8.

The directory label specifies for its drive whether password support is to be activated, and if date and time stamping for files is to be performed. The format of the directory label is shown below in Figure 2-2.

DR	NAME	TYPE	DL	S1	S2	RC	PASSWORD	TS1	TS2
00	01...	09...	12	13	14	15	16...	25.	29.

Figure 2-2. Directory Label Format

Table 2-8. Directory Label Field Definitions

<i>Field</i>	<i>Definition</i>
DR	drive code (0 - 16)
Name	directory label name
Type	directory label type
DL	directory label data byte  Bit 7 - enable password support Bit 6 - perform access time stamping Bit 5 - perform update time stamping Bit 4 - perform create time stamping Bit 0 - Directory Label exists (Bit references are right to left, relative to 0)
S1,S2,RC	reserved for future use
Password	8-byte password field (encrypted)
TS1	4-byte creation time stamp field
TS2	4-byte update time 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 can be used to identify a disk.



You can use the `DRV_SETLABEL` system call to create a directory label or update its fields. This system call can also assign a password to a directory label. The directory label password, if assigned, cannot be circumvented, whereas file password protection on a drive is an option controlled by the directory label. Thus, access to the directory label password provides the ability to bypass password protection on the drive.

**Note:** the file system provides no specific system call to read the directory label FCB directly. However, you can read the directory label data byte directly with the BDOS system call, `DRV_GETLABEL`. In addition, you can use the BDOS search system calls `F_SFIRST` and `F_SNEXT` to find a directory label. You can identify the directory label by a value of 32 (020H) in byte 0 of the directory FCB.

The XFCB is an extended FCB that can optionally be associated with a file in the directory. If present, it contains the file's password and password mode. The format of the XFCB is shown below in Figure 2-3.

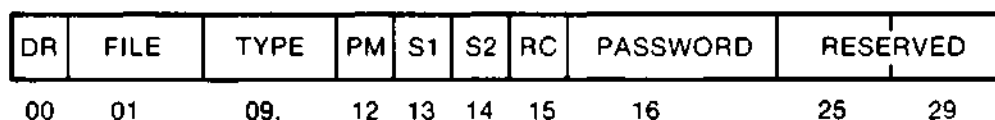


Figure 2-3. XFCB - Extended File Control Block

The fields in the XFCB are defined in Table 2-9:

**Table 2-9. XFCB Field Definitions**

<i>Field</i>	<i>Definition</i>
DR	drive code (0 - 16)
File	filename field
Type	filetype field
PM	password mode  Bit 7 - Read mode Bit 6 - Write mode Bit 5 - Delete mode (Bit references are right to left, relative to 0)
S1,S2,RC	reserved for system use
Password	8-byte password field (encrypted)
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 a file: with the F\_MAKE system call or the F\_WRITEXFCB system call. The F\_MAKE system call creates an XFCB if the calling process requests that a password be assigned to the created file. The F\_WRITEXFCB system call creates an XFCB when it is called to assign a password to an existing file. You can identify an XFCB in the directory by a value of 16 (010H) + N in byte 0 of the FCB, where N equals the user number.

## 2.7 File Passwords

There are two ways to assign passwords to a file: by the `F_MAKE` system call or by the `F_WRITEXFCB` system call. You can also change a file's password or password mode with the `F_WRITEXFCB` system call if you can supply the original password. Note that you cannot change a file's password or password mode if password protection for the drive is disabled by the directory label. However, even if you cannot supply a file's password, you can delete a file's XFCB, thereby removing its password protection, if password protection is disabled on the drive.

The Concurrent CP/M-86 BDOS provides password protection in one of three modes when password support is enabled by the directory label. Table 2-10 shows the difference in access level allowed to BDOS system calls when the password is not supplied.

Table 2-10. Password Protection Modes

<i>Mode</i>	<i>Access Level Allowed Without Password</i>
(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 process must supply the password to open the file. Processes 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 process must specify the password to delete or truncate the file, rename the file, or to modify the file's attributes. Thus, password protection in mode 1 implies mode 2 and 3 protection, and mode 2 protection implies mode 3 protection. All three modes require the user to specify the password to delete or truncate the file, rename the file, or to modify the file's attributes.

If a process supplies the correct password or the directory label disables password protection, then access to the BDOS system calls is the same as for a file that is not password-protected. In addition, the F\_SFIRST and F\_SNEXT system calls are not affected by file passwords. The following BDOS system calls test for passwords.

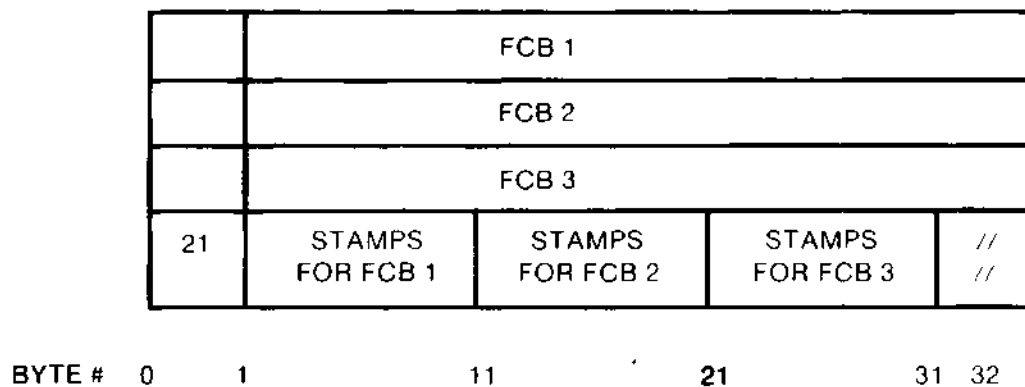
DRV\_SETLABEL  
F\_ATTRIB  
F\_DELETE  
F\_OPEN  
F\_RENAME  
F\_WRITEXFCB  
F\_TRUNCATE

The BDOS maintains file passwords in the XFCB and directory label in encrypted form. To make a BDOS system call for a file that requires a password, a process must place the password in the first eight bytes of the current DMA, or make it the default password with the F\_PASSWD system call, before making the system call.

**Note:** the BDOS maintains the assigned default password for each process. Processes inherit the default password of their parent process. You can set a given TMP's default password using the SET command; all programs loaded by this TMP inherit the same default password.

## 2.8 File Date and Time Stamps: SFCBs

The Concurrent CP/M-86 file system uses a special type of directory entry called an SFCB to record date and time stamps for files. When a directory has been initialized for date and time stamping, SFCBs reside in every fourth position of the directory. Each SFCB maintains the date and time stamps for the previous three directory entries, as shown in Figure 2-4.

**Figure 2-4. 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 entry in the directory record.

The SFCB itself contains five fields. The first field is a single byte containing the value 021H; 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 2-5 shows the detail of the SFCB subfields.

**Figure 2-5. SFCB Subfields**

An SFCB subfield only contains valid information if its corresponding FCB in the directory record is an extent zero FCB. This FCB is a file's first directory entry. For password protected files, the SFCB subfield also contains the password mode of the file; the password mode field is zero for files without password protection. You can read SFCBs by making `F_SFIRST` and `F_SNEXT` system calls. In addition, you can make a `F_TIMEDATE` system call to retrieve the date and time stamps and password mode of a specified file. Refer to the `T_GET` system call definition in Section 6 for the description of the format of a date and time stamp field.

Concurrent CP/M-86 supports three kinds of file stamping: create, access, and update. Create stamps record when the file was created, access stamps record when the file was last opened, and update stamps record the last time the file was modified. Create and access stamps share the same field. As a result, file access stamps overwrite any create stamps.

The directory label of a properly initialized disk determines the type of date and time stamping for files on the drive. The INITDIR utility initializes a directory for date and time stamping by placing an SFCB in every fourth directory entry. Disks not initialized in this way cannot support date and time stamping. In addition, date and time stamping is not performed if the disk's directory label is absent or does not specify date and time stamping, or if the disk is Read-Only.

Note that the directory label is also time stamped, but these stamps are not made in an SFCB; time stamp fields in the last eight bytes of the directory label show when it was created and last updated. Access stamping is not supported for directory labels.

The BDOS file system uses the system date and time when it records a date and time stamp. This value is maintained in a field in the SYSDAT part of the System Data Segment. The DATE utility sets the system time and date (refer to the *Concurrent CP/M-86 User's Guide* for details of using DATE).

## 2.9 File Open Modes

The file system provides three different modes for opening files. They are defined below.

### Locked Mode

A process can open a file in Locked mode only if the file is not currently opened by another process. Once open in Locked mode, no other process can open the file until it is closed. Thus, if a process successfully opens a file in Locked mode, that process owns the file until the file is closed or the process terminates. Files opened in Locked mode support read and write operations unless the file is a Read-Only file (attribute T1' set) or the file is password-protected in Write mode, and the process issuing the F\_OPEN call cannot supply the password. In both of these cases, the BDOS allows only read operations to the file.

**Note:** Locked mode is the Default mode for opening files under Concurrent CP/M-86.

### Unlocked Mode

A process can open a file in Unlocked mode if the file is not currently open, or if another process has already opened the file in Unlocked mode. This mode allows more than one process to open the same file. Files opened in Unlocked mode support read and write operations unless the file is a Read-Only file (attribute T1' set) or the file is password-protected in Write mode and the process issuing the F\_OPEN call cannot supply the password.

When opening a file in Unlocked mode, a process must reserve 35 bytes in the FCB because the F\_OPEN system call returns a 2-byte value called the File ID in the R0 and R1 bytes of the FCB. The File ID is a required parameter for the F\_LOCK and F\_UNLOCK system calls. These BDOS system calls work only for files opened in Unlocked mode.

### Read-Only Mode

A process can open a file in Read-Only mode if the file is not currently opened by another process or if another process has opened the file in Read-Only mode. This mode allows more than one process to open the same file for Read-Only access.

The F\_OPEN system call performs the following steps for files opened in Locked or Read-Only mode. If the current user number is nonzero, and the file to be opened does not exist under the current user number, the F\_OPEN system call searches the user zero directory for the file. If the file exists under user zero and has the system attribute T2' set, the BDOS opens the file under user zero. The open mode is automatically forced to Read-Only when this is done.

The F\_OPEN system call also performs the following action for files opened in Locked mode when the current user number is zero. If the file exists under user zero and has the system T2' and Read-Only (T1') attributes set, the open mode is automatically set to Read-Only. The Read-Only attribute controls whether a user-zero process and processes on other user numbers can concurrently open a user-zero system file when each process opens the file in the default Locked mode. If the Read-Only attribute is set, all processes open the file in Read-Only mode and the BDOS allows concurrent access of the file. However, if the Read-Only attribute is reset, the user-zero process opens the file in Locked mode. This prevents sharing the file with other processes.



The `F_OPEN` and `F_MAKE` system calls use FCB interface attributes `F5'` and `F6'` to specify the open mode. The interface attribute definitions for these functions are listed in Table 2-7.

**Note:** the `F_MAKE` system call does not allow opening the file in Read-Only mode.

## 2.10 File Security

In general, the security measures implemented by the file system prevent accidental collisions between running processes. It is not possible to provide total security under Concurrent CP/M-86 because the file system maintains file allocation information in open FCBs in the user's memory region, and Concurrent CP/M-86 does not require memory protection. However, the file system is designed to ensure that multiple processes can share the same file system without interfering with each other by

- performing checksum verification of open FCBs.
- monitoring all open files and locked records via the system Lock List.

The BDOS validates the checksum of user FCBs before all I/O operations to protect the integrity of the file system from corrupted FCBs. The `F_OPEN` and `F_MAKE` system calls compute and assign checksums to FCBs. The `F_READRAND`, `F_READ`, `F_WRITERAND`, `F_WRITEZF`, `F_WRITE`, `F_LOCK`, and `F_UNLOCK` system calls subsequently verify and recompute the checksums when they change the FCB. The `F_CLOSE` system call also verifies FCB checksums. Note that FCB verification by these system calls can be disabled (see Section 2.12), but Concurrent CP/M-86's file security is reduced when this is done. If the BDOS detects an FCB checksum error, it does not perform the requested command. Instead, it either returns to the calling process with an error code, or if the system call is `F_CLOSE` and the BDOS Error mode is in the default state (see Section 2.18), it terminates the calling process with an error message.

Concurrent CP/M-86 uses a system data structure, called the Lock List, to manage file opening and record locking by running processes. Each time a process opens a file or locks a record successfully, the file system allocates an entry in the system Lock List to record the fact. The file system uses the following information to

- prevent a process from deleting, truncating, renaming, or updating the attributes of another process's open file.
- prevent a process from opening a file currently opened by another process, unless both processes open the file in unlocked or Read-Only mode.
- prevent a process from resetting a drive on which another process has an open file.
- prevent a process from reading, writing, or locking a record currently locked by another process. Refer to Section 2.14 for more information on record locking and unlocking.

The file system only verifies whether another process has the FCB-specified file open for the following file-access system calls: F\_OPEN, F\_MAKE, F\_DELETE, F\_RENAME, F\_ATTRIB, and F\_TRUNCATE. For file-access system calls that require an open FCB, the FCB checksum controls whether the calling process can use the FCB. By definition, a valid FCB checksum implies that the file has been successfully opened and an entry for the file resides in the system Lock List.

The most common way a process releases a lock entry for an open file is by closing the file. A close operation is permanent if it causes the removal of the file's open lock list entry. The file system invalidates the FCB checksum field on permanent close operations to prevent continued open file operations with the FCB.

However, not all close operations are permanent. For example, if a process makes multiple F\_OPEN or F\_MAKE calls to an open file, a matching number of F\_CLOSE calls must be made before the file system permanently closes the file. Of course, if you only open a file once, a single close operation permanently closes the file. In addition, a process can optionally make partial F\_CLOSE calls to a file by setting interface attribute F5'. A partial close operation does not affect the open state of a file. In the above example, a partial close operation would not count against an F\_OPEN or F\_MAKE call. A partial close operation simply updates the directory to reflect the current state of the file.

As a general rule, under Concurrent CP/M-86 a process should close files as soon as it no longer needs them, even if it has not modified them. While a process has a file open, access by other processes to the file is restricted. For example, after a process has opened a file in Locked mode, the file cannot be opened by other processes until the file is closed or the process terminates.

Furthermore, space in the system Lock List is limited. If a process attempts to open a file and no space remains in the system Lock List, or if the process exceeds the open file limit, the BDOS denies the open request and usually terminates the calling process. You can change the way the file system handles this error by making a `F_ERRMODE` system call. Note that the size of the system Lock List and the process open file limit are GENCCPM parameters.

There are several other situations where the file system removes open file entries from the system Lock List for a process. For example, if a process makes a `F_DELETE` call for a file it has open in Locked mode, the file system deletes the file and also purges the file's entry from the system Lock List. Deleting an open file is not recommended under Concurrent CP/M-86 but it is supported for files opened in Locked mode to provide compatibility with software written under earlier releases of MP/M™ and CP/M. The file system does not allow deletion of a file opened in Unlocked or Read-Only mode.

To ensure that the process does not use the open FCB corresponding to the deleted file, the file system subsequently checks all open FCBs for the process. Each open FCB is checked the next time it is used with a file-access system call that requires an open FCB. If a Lock List entry exists for the file, the BDOS allows the operation to proceed; if not, it indicates that the file has been purged and the file system returns an FCB checksum error.

The file system performs this verification of a process's open FCBs whenever it purges an open file entry from the system Lock List. The following list describes these situations:

- A process makes a `F_ATTRIB`, `F_DELETE`, `F_RENAME`, or `F_TRUNCATE` system call to a file it has open in Locked mode. These operations cannot be performed on a file open in Unlocked or Read-Only mode.

- A process issues a DRV\_FREE call for a drive on which it has an open file.
- The BDOS detects a change in media on a drive that has open files. This is a special case because a process cannot control the occurrence of this situation, and because it can impact more than one process. Refer to Section 2.17 for more details on this situation.

Open FCB verification can affect performance because each verification operation requires a directory search operation. In general, you should avoid such situations when creating new programs for Concurrent CP/M-86.

## 2.11 Extended File Locking

Extended file locking enables a Concurrent CP/M-86 process to maintain a lock on a file after the file is permanently closed. This facility allows a process to set the attributes, delete, rename, or truncate a file without interference from other processes. In addition, this technique avoids the problems associated with using these system calls on open files (see Section 2.10).

A process can also reopen a file with an extended lock and continue open file processing. To illustrate how extended file locking might be used, a process can close an open file, rename the file, reopen the file under its new name, and continue with file operations without ever losing the file's Lock List item and control over the file.

A process can only specify extended file locking for a file it has opened in Locked mode. To extend a file's lock, set interface attribute F6' when closing the file. The F\_CLOSE system call interrogates this attribute only when it is closing a file permanently. Thus, interface attribute F5', signifying a partial close, must be reset when the F\_CLOSE call is made. In addition, the close operation must be permanent. If a process has opened a file N times, the F\_CLOSE system call ignores the F6' attribute until the file is closed for the Nth time.

Note that the access rules for a file with an extended lock are identical to the rules for a file open in Locked mode.

To maintain an extended file lock through a `F_ATTRIB`, `F_RENAME`, or `F_TRUNCATE` system call, set interface attribute `F5'` of the referenced FCB when making the call. The BDOS honors this attribute only if the file has been closed with an extended lock. Setting attribute `F5'` also maintains an extended file lock for the `F_DELETE` system call, but setting this attribute also changes the nature of the delete operation to an XFCB-only delete. If successful, all four of these system calls delete a file's extended lock item if they are called with attribute `F5'` reset. However, the extended lock item is not deleted if they return with an error code.

You can make a `F_OPEN` call to resume record operations on a file with an extended lock. Note that you can also change the open mode when you reopen the file. The following example illustrates the use of extended locks.

1. Open file `EXLOCK.TST` in Locked mode.
2. Perform read and write operations on the file `EXLOCK.TST` using the open FCB.
3. Close file `EXLOCK.TST` with interface attribute `F6'` set to retain the file's lock item.
4. Use the `F_RENAME` system call to change the name of the file to `EXLOCK.NEW` with interface attribute `F5'` set to retain the file's extended lock item.
5. Reopen the file `EXLOCK.NEW` in Locked mode.
6. Perform read and write operations on the file `EXLOCK.NEW`, using the open FCB.
7. Close file `EXLOCK.NEW` again with interface attribute `F6'` set to retain the file's lock item.
8. Set the Read-Only attribute and release the file's lock item by making a `F_ATTRIB` system call with interface attribute `F5'` reset.

At this point, the file `EXLOCK.NEW` becomes available for access by another process.

## 2.12 Compatibility Attributes

Compatibility attributes provide a mechanism to modify some of the Concurrent CP/M-86 file security rules for specific command files. Concurrent CP/M-86 includes this facility because some programs developed under earlier Digital Research operating systems do not run properly under Concurrent CP/M-86. Most of the problems encountered by these programs occur because they were designed for single-tasking operating systems where file security is not required. For example, a program might close a file and then continue reading and writing to the file. Under CP/M-86, this does not cause a problem. However, under Concurrent CP/M-86, the file system intercepts open file operations with a deactivated FCB to ensure the integrity of the file system. With compatibility attributes, you have a tool for dealing with these kinds of situations.

You should only use compatibility attributes with existing programs that run properly under CP/M or CP/M-86. Do not use compatibility attributes with new programs you develop under Concurrent CP/M-86.

Compatibility attributes are defined as file attributes F1' through F4' of program (CMD) files. You can use the Concurrent CP/M-86 SET utility to set these file attributes from the command line. However, setting a command file's compatibility attributes has no affect unless the GENCCPM COMPATMODE option has been selected during system generation. If this has been done, the P\_CLI system call interrogates file attributes F1' through F4' of the command file during program loading and modifies the Concurrent CP/M-86 file security rules for the loaded program.

The Concurrent CP/M-86 BDOS defines the compatibility attributes as shown in Table 2-11.

Table 2-11. Compatibility Attribute Definitions

<i>Attribute</i>	<i>Definition</i>
F1'	<p>Modify the rules for Locked mode.</p> <p>When a process running with F1' set opens a file in Locked mode, it can perform read and write operations to the file as normal. However, to other processes on the system, it appears as if the file was opened in Read-Only mode. Thus, another process running with F1' set, can open the same file in Locked mode and also perform write operations to the file. In addition, if a process with F1' reset attempts to open the file in Locked or Read-Only mode, the open attempt is allowed but the open mode is forced to Read-Only. Furthermore, write operations are not allowed when the process has F1' reset.</p> <p>This compatibility mode is designed to allow multiple copies of the same program to run concurrently, even though the program might make read and write calls to a common file that it has opened in Locked mode. In addition, this compatibility mode allows other programs not in this compatibility mode to access the file on a Read-Only basis. Note that record locking is not supported for this modified open mode. In addition, to be safe, make all static files such as program and help files Read-Only if you use this compatibility attribute.</p> <p>There is an alternative to using this attribute if a program only makes read calls to the common file. By placing the file under User 0 with the SYS and Read-Only attributes set, you force the open mode to Read-Only when the file is opened in Locked mode.</p>

Table 2-11. (continued)

<i>Attribute</i>	<i>Definition</i>
F2' Change F_CLOSE to partial close.	<p>Processes running with F2' set, only make partial F_CLOSE system calls. This attribute is intended for programs that close a file to update the directory but continue to use the file. A side effect of this attribute is that files opened by a process are not released from the system Lock List until the process terminates. When using this attribute, it might be necessary to set the system Lock List parameters to higher values when you generate a system with GENCCPM.</p>
F3' Ignore close checksum errors.	<p>This attribute changes the way the F_CLOSE system call handles Close Checksum errors. Normally, the file system prints an error message on the console and terminates the calling process. However, if this attribute is set, the F_CLOSE system call ignores the checksum error and performs the close operation. This interface attribute is intended for programs that modify an open FCB before closing a file.</p>
F4' Disable FCB Checksum verification for read and write operations.	<p>Setting this attribute also sets attributes F2' and F3'. This attribute is intended for programs that modify open FCBs during read and write operations. Use this attribute very carefully, and only with software known to work, because it effectively disables Concurrent CP/M-86's file security.</p>



Use the Concurrent CP/M-86 SET utility to specify the combination of compatibility attributes you want set in the program's command file. For example,

```
A>SET filespec [f1=on]
A>SET filespec [f1=on,f3=on]
A>SET filespec [f4=on]
```

If you have a program that runs under CP/M or CP/M-86 but does not run properly under Concurrent CP/M-86, use the following guidelines to select the proper compatibility attributes for the program.

- If the program ends with the "File Currently Opened" message when multiple copies of the program are run, set compatibility attribute F1', or place all common static files under User 0 with the SYS and Read-Only attributes set.
- If the program terminates with the message "Close Checksum Error", set compatibility attribute F3'.
- If the program terminates with an I/O error, try running the program with attribute F2' set. If the problem persists, then try attribute F4'. Use attribute F4' only as a last resort.

## 2.13 Multisector I/O

The BDOS file system provides the capability to read or write multiple 128-byte records in a single BDOS system call. This multisector facility can be visualized as a BDOS burst mode, enabling a process to complete multiple I/O operations without interference from other running processes. In addition, the BDOS file system bypasses, when possible, all intermediate record buffering during multisector I/O operations. Data is transferred directly between the calling process's memory and the drive. The BDOS also informs the XIOS when it is reading or writing multiple physical records on a drive. The XIOS can use this information to further optimize the I/O operation resulting in even better performance. As a result, the use of this facility in an application program can improve its performance and also enhance overall system throughput, particularly when performing sequential I/O.

The number of records that can be transferred with multisector I/O ranges from 1 to 128. This value, called the BDOS Multisector Count, can be set by the F\_MULTISEC system call. The P\_CLI system call sets the Multisector Count to one when it initiates a transient program for execution. Note that the greatest potential performance increases are obtained when the Multisector Count is set to 128. Of course, this requires a 16K buffer. The Concurrent CP/M-86 PIP utility performs its sequential I/O with a Multisector Count of 128.

The Multisector Count determines the number of operations to be performed by the following BDOS system calls:

- F\_READ and F\_WRITE system calls
- F\_READRAND, F\_WRITERAND, and F\_WRITEZF
- F\_LOCK and F\_UNLOCK

If the Multisector Count is N, calling one of the above system calls is equivalent to making N system calls. With the exception of disk I/O errors encountered by the XIOS, if an error interrupts a multisector read or write operation, the file system returns the number of 128-byte records successfully transferred in register AH. Section 2.14 describes how the Multisector Count affects the F\_LOCK and F\_UNLOCK system calls.

## 2.14 Concurrent File Access

Concurrent CP/M-86 supports two open modes, Read-Only and Unlocked, which allow concurrently running processes to access common files for record operations. The Read-Only open mode allows multiple processes to read from a common file, but processes cannot write to a file open in this mode. Thus, files remain static when they are opened in Read-Only mode. The Unlocked open mode is more complex because it allows multiple processes to read and write records to a common file. As a result, Unlocked mode has some important differences from the other open modes.

When a process opens a file in Unlocked mode, the file system returns a 2-byte field called the File ID in the R0 and R1 bytes of the FCB. The File ID is a required parameter of Concurrent CP/M-86's record locking system calls, `F_LOCK` and `F_UNLOCK`, which are only supported for files open in Unlocked mode. Note that these system calls return a successful error code if they are called for files opened in Locked mode. However, they perform no action in this case, because, by definition, the calling process has the entire file locked.

The `F_LOCK` and `F_UNLOCK` system calls allow a process to establish and release temporary ownership to particular records within a file. You must set the FCB Random Record field and place the File ID in the first two bytes of the current DMA buffer before making these calls. The file system locks and unlocks records in units of 128 bytes, which is the standard Concurrent CP/M-86 record size. The number of records locked or unlocked is controlled by the BDOS Multisector count, which can range from 1 to 128 (see Section 2.13). In order to simplify the discussion of record locking and unlocking, the following paragraphs assume the Multisector count is one. However, as discussed later in this section, the more general case of multiple record locking and unlocking is a simple extension of the single record case.

The `F_LOCK` system call supports two types of lock operations: exclusive locks and shared locks. Interface attribute `F5'` specifies the type of lock. `F5' = 0` requests an exclusive lock; `F5' = 1` requests a shared lock. If a process locks a record with an exclusive lock, other processes cannot read, write, or lock the record. The locking process, however, can access the record with no restrictions. You should use this type of lock when exclusive control over a record is required.

If a process locks a record with a shared lock, other processes cannot write to the record or make an exclusive lock of the record. However, other processes are allowed to read the record and make their own shared locks on the record. No process, including the locking process, can write to a record with a shared lock. Shared locks are useful when you want to ensure that a record does not change, but you want to allow other processes to read the record.

The `F_LOCK` system call also lets you change the lock of a record if there is no conflict. For example, you can convert an exclusive lock into a shared lock with no restrictions. On the other hand, a process cannot convert a record's shared lock to an exclusive lock if another process has a shared lock on the record.

The `F_LOCK` system call has another option, specified by interface attribute `F6'`, which controls whether a record must exist in order to be locked. If you make a `F_LOCK` system call with `F6' = 0`, the file system returns an error code if the specified record does not exist within the file. Setting `F6'` to 1 requests a logical lock operation. Logical lock operations are only limited by the maximum Concurrent CP/M-86 file size of 32 megabytes, which corresponds to a maximum Random Record Number of 262,143. You can use logical locks to control extending a shared file.

The `F_UNLOCK` system call is similar to the `F_LOCK` call except that it removes locks instead of creating them. There are few restrictions on unlock operations. Of course a process can only remove locks that it has made. The `F_UNLOCK` system call has one option, controlled by interface attribute `F5'`. If `F5'` is set to one, the `F_UNLOCK` system call removes all locks for the file made by the calling process. Otherwise, it removes the locks specified by the Random Record field and the BDOS Multisector Count. Note that the `F_CLOSE` system call also removes all locks for a file on permanent close operations.

If the BDOS Multisector Count is greater than one, the `F_LOCK` and `F_UNLOCK` system calls perform multiple record locking or unlocking. In general, multiple record locking and unlocking can be viewed as a sequence of  $N$  independent operations, where  $N$  equals the Multisector Count. However, if an error occurs on any record within the sequence, no locking or unlocking is performed. For example, both `F_LOCK` and `F_UNLOCK` perform no action and return an error code if the sum of the FCB Random Record Number and the BDOS Multisector Count is greater than 262,144. As another example, the `F_LOCK` system call also returns an error code if another process has an exclusive lock on any record within the sequence.

When a process makes a `F_LOCK` system call, the file system allocates a new entry in the system Lock List to record the lock operation and associate it with the calling process. A corresponding `F_UNLOCK` system call removes the locked entry from the list. While the lock entry exists in the system Lock List, the file system enforces the restrictions implied by the lock item.

Because each lock item includes a record count field, a multiple lock operation normally results in the creation of a single new entry. However, if the file system must split an existing lock entry to satisfy the lock operation, an additional entry is required. Similarly, an unlock operation can require the creation of a new entry if a split is needed. Thus, in the worst case, a lock operation can require two new lock entries and an unlock operation can require one. Note that lock item splitting can be avoided by locking and unlocking records in consistent units.

These considerations are important because the Lock List is a limited resource under Concurrent CP/M-86. The file system performs no action and returns an error code if insufficient available entries exist in the system Lock List to satisfy the lock or unlock request. In addition, the number of lock items a single process is allowed to consume is a GENCCPM parameter. The file system also returns an error code if this limit is exceeded.

The file system performs several special operations for read and write system calls to a file open in Unlocked mode. These operations are required because the file system maintains the current state of an open file in the calling process's FCB. When multiple processes have the same file open, FCBs for the same file exist in each process's memory. To ensure that all processes have current information, the file system updates the directory immediately when an FCB for an unlocked file is changed. In addition, the file system verifies error situations such as end-of-file, or reading unwritten data with the directory before returning an error. As a result, read and write operations are less efficient for files open in Unlocked mode when compared to equivalent operations for files opened in Locked mode.

## 2.15 File Byte Counts

Although the logical record size of Concurrent CP/M-86 is restricted to 128 bytes, the file system does provide a mechanism to store and retrieve a byte count for a file. This facility can identify the last byte of the last record of a file. The BDOS Compute File Size function returns the last Random Record Number, + 1, of the last record of a file.

The `F_ATTRIB` system call can set a file's byte count. This is an option controlled by interface attribute `F6'`. Conversely, the `F_OPEN` system call can return a file's byte count to the `CR` field of the `FCB`. The `F_SFIRST` and `F_SNEXT` system calls also return a file's byte count. These system calls return the byte count in the `CS` field of the `FCB` returned in the current DMA buffer.

Note that the file system does not access or update the byte count value in BDOS read or write system calls. However, the `F_MAKE` system call does set the byte count value to zero when it creates a file in the directory.

## 2.16 Record Blocking and Deblocking

Under Concurrent CP/M-86, the logical record size for disk I/O is 128 bytes. This is the basic unit of data transfer between the operating system and running processes. However, on disk, the record size is not restricted to 128 bytes. These records, called physical records, can range from 128 bytes to 4K bytes in size. Record blocking and deblocking is required on systems that support drives with physical record sizes larger than 128 bytes.

The process of building up physical records from 128-byte logical records is called record blocking. This process is required in write operations. The reverse process of breaking up physical records into their component 128-byte logical records is called record deblocking. This process is required in read operations. Under Concurrent CP/M-86, record blocking and deblocking is normally performed by the BDOS.

Record deblocking implies a read-ahead operation. For example, if a process reads a logical record that resides at the beginning of a physical record, the entire physical record is read into an internal buffer. Subsequent BDOS read calls for the remaining logical records access the buffer instead of the disk. Conversely, record blocking results in the postponement of physical write operations but only for data write operations. For example, if a transient program makes a BDOS write call, the logical record is placed in a buffer equal in size to the physical record size. The write operation on the physical record buffer is postponed until the buffer is needed in another I/O operation. Note that under Concurrent CP/M-86, directory write operations are never postponed.

Postponing physical record write operations has implications for some application programs. For programs that involve file updating, it is often critical to guarantee that the state of the file on disk parallels the state of the file in memory after an update operation. This is only an issue on drives where physical write operations are postponed because of record blocking and deblocking. If the system should crash while a physical buffer is pending, data would be lost. To prevent this loss of data, the `F_FLUSH` system call can be called to force the write of any pending physical buffers associated with the calling process.

**Note:** the file system discards all pending physical data buffers when a process terminates. However, the file system automatically makes a `F_FLUSH` call in the `F_CLOSE` system call. Thus, it is sufficient to make a `F_CLOSE` system call to ensure that all pending physical buffers for that file are written to the disk.

## 2.17 Reset, Access, and Free Drive

The BDOS system calls `DRV_ALLRESET`, `DRV_RESET`, `DRV_ACCESS`, and `DRV_FREE` allow a process to control when to reinitialize a drive directory for file operations. This process of initializing a drive's directory is called logging-in the drive.

When you start Concurrent CP/M-86, all drives are initialized to the reset state. Subsequently, as processes reference drives, the file system automatically logs them in. Once logged-in, a drive remains in the logged-in state until it is reset by the `DRV_ALLRESET` or `DRV_RESET` system calls or a media change is detected on the drive. If the drive is reset, the file system automatically logs in the drive again the next time a process references it. The file system logs in a drive immediately when it detects a media change on the drive.

Note that the `DRV_ALLRESET` and `DRV_RESET` system calls have similar effects except that the `DRV_ALLRESET` system call affects all drives on the system. You can specify the combination of drives to reset with the `DRV_RESET` system call.

Logging-in a drive consists of several steps. The most important step is the initialization of the drive's allocation vector. The allocation vector records the allocation and deallocation of data blocks to files, as files are created, extended, deleted and truncated. Another function performed during drive log-in is the initialization of the directory checksum vector. The file system uses the checksum vector to detect media changes on a drive. Note that permanent drives, which do not support media changes, might not have checksum vectors.

Under Concurrent CP/M-86, the DRV\_RESET operation is conditional. The file system cannot reset a drive for a process if another process has an open file on the drive. However, the exact action taken by a DRV\_RESET operation depends on whether the drive to be reset is permanent or removable.

Concurrent CP/M-86 determines whether a drive is permanent or removable by interrogating a bit in the drive's Disk Parameter Block (DPB) in the XIOS. A high-order bit of 1 in the DPB Checksum Vector Size field designates the drive as permanent. A drive's Removable or Nonremovable designation is critical to the reset operation described below.

The BDOS first determines whether there are any files currently open on the drive to be reset. If there are none, the reset takes place. If there are open files, the action taken by the reset operation depends on whether the drive is removable and whether the drive is Read-Only or Read-Write. Note that only the DRV\_SETRO system call can set a drive to Read-Only. Following log-in, a drive is always Read-Write.

If the drive is a permanent drive and if the drive is not Read-Only, the reset operation is not performed, but a successful result is returned to the calling process.

However, if the drive is removable or set to Read-Only, the file system determines whether other processes have open files on the drive. If they do, then it denies DRV\_RESET operation and returns an error code to the calling process.

If all the open files on a removable drive belong to the calling process, the process is said to own the drive. In this case, the file system performs a qualified reset on the drive and returns a successful result. This means that the next time a process accesses this drive, the BDOS performs the log-in operation only if it detects a media change on the drive. The logic flow of the drive reset operation is shown in Figure 2-6.



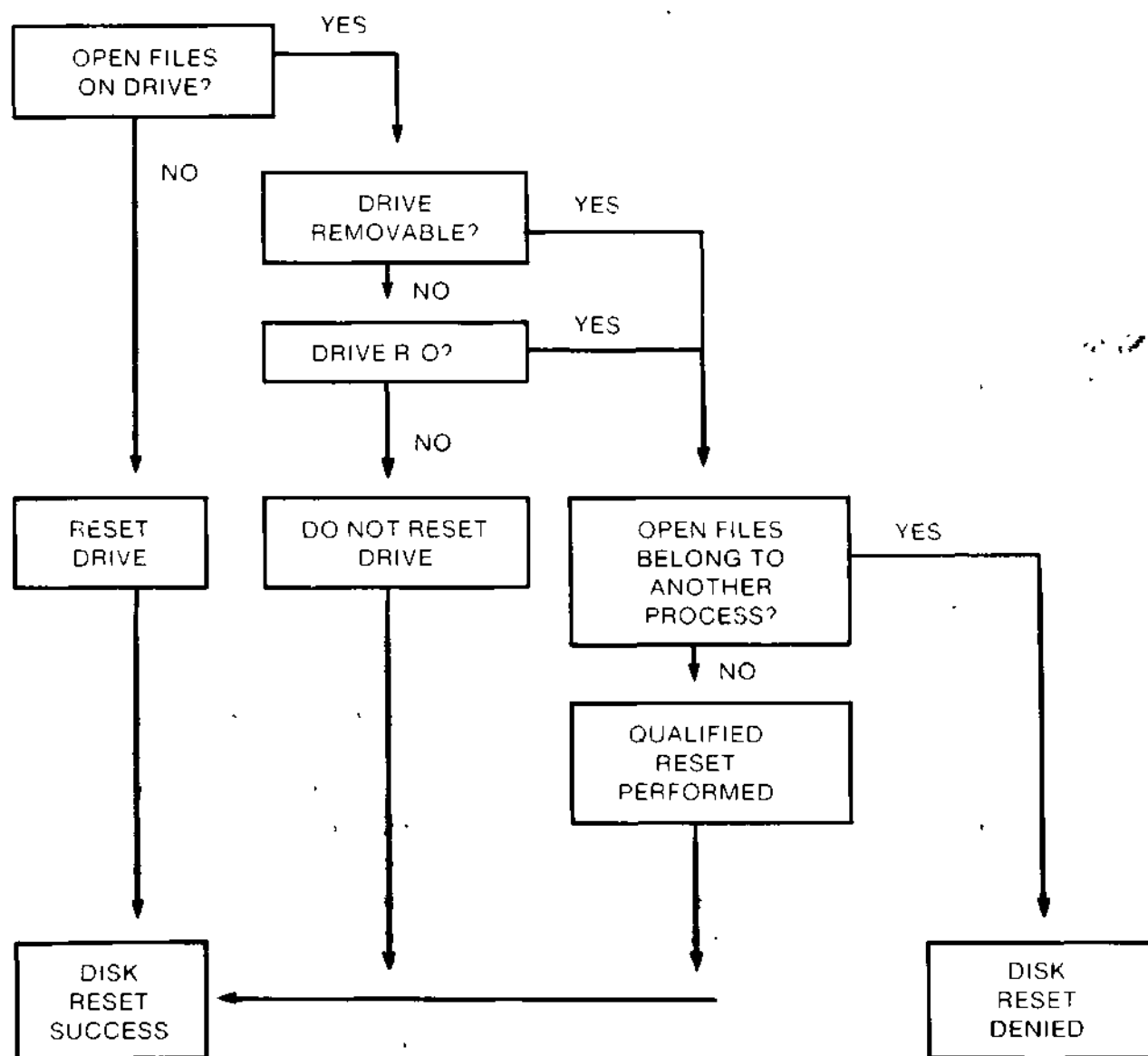


Figure 2-6. Disk System Reset

If the BDOS detects a media change on a drive after a qualified reset, it purges all open files on the drive from the system Lock List and subsequently verifies all open FCBs in file operations for the owning process (refer to Section 2.10 for details of FCB verification).

In all other cases where the BDOS detects a media change on a drive, the file system purges all open files on the drive from the system Lock List, and flags all processes owning a purged file for automatic open FCB verification.

**Note:** if a process references a purged file with a BDOS command that requires an open FCB, the file system returns to the process with an FCB checksum error.

The primary purpose of the drive reset functions is to prepare for a media change on a drive. Because a drive reset operation is conditional, it allows a process to test whether it is safe to change disks. Thus, a process should make a successful drive reset call before prompting the user to change disks. In addition, you should close all your open files on the drive, particularly files you have written to, before prompting the user to change disks. Otherwise, you might lose data.

The DRV\_ACCESS and DRV\_FREE system calls perform special actions under Concurrent CP/M-86. The DRV\_ACCESS system call inserts a dummy open file item into the system Lock List for each specified drive. While that item exists in the system Lock List, no other process can reset the drive. The DRV\_FREE system call purges the Lock List of all items, including open file items, belonging to the calling process on the specified drives. Any subsequent reference to those files by a BDOS system call requiring an open FCB results in a FCB checksum error return.

The DRV\_FREE system call has two important side effects. First of all, any pending blocking/deblocking buffers on a specified drive that belong to the calling process 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 before making this system call.

The `DRV_SETRO` system call is also conditional under Concurrent CP/M-86. The file system does not allow a process to set a drive to Read-Only if another process has an open file on the drive. This applies to both removable and permanent drives.

A process can prevent other processes from resetting a Read-Only drive by opening a file on the drive or by issuing a `DRV_ACCESS` call for the drive and then making a `DRV_SETRO` system call. Executing `DRV_SETRO` before the `F_OPEN` or `DRV_ACCESS` call leaves a window in which another process could set the drive back to Read-Write. While the open file or dummy item belonging to the process resides in the system Lock List, no other process can reset the drive to take it out of Read-Only status.

## 2.18 BDOS Error Handling

The Concurrent CP/M-86 file system has an extensive error handling capability. When an error is detected, the BDOS responds in one of three ways:

1. It can return to the calling process with return codes in the AX register identifying the error.
2. It can display an error message on the console and terminate the process.
3. It can display an error message on the console and return an error code to the calling process, as in method 1.

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

The BDOS Error mode, which has three states, determines how the file system handles physical and extended errors. In the default state, the BDOS displays the error message and terminates the calling process (method 2). In Return Error mode, the BDOS returns control to the calling process with the error identified in the AX register (method 1). In Return and Display Error mode, the BDOS returns control to the calling process with the error identified in the AX register and also displays the error message at the console (method 3).

While both return modes protect a process from termination because of a physical or extended error, the Return and Display mode also allows the calling process to take advantage of the built-in error reporting of the file system. Physical and extended errors are displayed on the console in the following format:

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

where d is the name of the drive selected when the error condition occurs; error message identifies the error; nn is the BDOS function number, and filename.typ identifies the file specified by the BDOS function. If the BDOS function did not involve an FCB, the file information is omitted.

The following tables detail BDOS physical and extended error messages.

Table 2-12. BDOS Physical Errors

<i>Message</i>	<i>Meaning</i>
Disk I/O	The "Disk I/O" error results from an error condition returned to the BDOS from the XIOS module. The file system makes XIOS read and write calls to execute BDOS file-access system calls. If the XIOS read or write routine detects an error, it returns an error code to the BDOS, causing this error message.
Invalid Drive	The "Invalid Drive" error also results from an error condition returned to the BDOS from the XIOS module. The BDOS makes an XIOS Select Disk call before accessing a drive to perform a requested BDOS function. If the XIOS does not support the selected disk, it returns an error code resulting in this error.
Read/Only File	The BDOS returns the "Read/Only File" error message when a process attempts to write to a file with the R/O attribute set.
Read/Only Disk	The BDOS returns the "Read/Only Disk error" message when a process makes a write operation to a disk that is in Read-Only status. A drive can be placed in Read-Only status explicitly with the DRV_SETRO system call.

Table 2-13. BDOS Extended Errors

<i>Message</i>	<i>Meaning</i>
File Opened in Read/Only Mode	<p>The BDOS returns the "File Opened in Read/Only Mode" error message when a process attempts to write to a file opened in Read-Only mode. A process can open a file in Read-Only mode explicitly by setting FCB interface attribute F6'. In addition, if a process opens a file in Locked mode, the file system automatically forces the open mode to Read-Only mode when:</p> <ul style="list-style-type: none"> <li>■ the current user number is zero and the process opens a file with the Read-Only and SYS attributes set.</li> <li>■ the current user number is not zero and the process opens a user zero file with the SYS attribute set.</li> </ul> <p>The BDOS also returns this error if a process attempts to write to a file that is password-protected in Write mode, and it did not supply the correct password when it opened the file.</p>
File Currently Open	<p>The BDOS returns the "File Currently Open" error message when a process attempts to delete, rename, or modify the attributes of a file opened by another process. The BDOS also returns this error when a process attempts to open a file in a mode incompatible with the mode in which the file was previously opened by another process or by the calling process.</p>
Close Checksum Error	<p>The BDOS returns the "Close Checksum Error" message when the BDOS detects a checksum error in the FCB passed to the file system with a F_CLOSE call.</p>
Password Error	<p>The BDOS returns the "Password Error" message when passwords are required and the file password is not supplied or is incorrect.</p>

Table 2-13. (continued)

<i>Message</i>	<i>Meaning</i>
File Already Exists	The BDOS returns the "File Already Exists" error message for the F_MAKE and F_RENAME system calls when the BDOS detects a conflict on filename and filetype.
Illegal ? in FCB	The BDOS returns the "Illegal ? in FCB" error message when the BDOS detects a ? character in the filename or filetype of the passed FCB for the F_ATTRIB, F_OPEN, F_RENAME, F_TIMEDATE, F_WRITEFCB, F_TRUNCATE, and F_MAKE system calls.
Open File Limit Exceeded	The BDOS returns the "Open File Limit Exceeded" error message when a process exceeds the process file lock limit specified by GENCCPM. The F_OPEN, F_MAKE, and DRV_ACCESS system calls can return this error.
No Room in System Lock List	The BDOS returns the "No Room in System Lock List" error message when no room for new entries exists within the system Lock List. The F_OPEN, F_MAKE, and DRV_ACCESS system calls can return this error.

The following paragraphs describe the error return code conventions of the file system calls. Most file system calls fall into three categories in regard to return codes; they return an error code, a directory code, or an error flag. The error conventions let programs written for CP/M-86 run without modification.

The following BDOS system calls return a logical error in register AL:

F\_LOCK  
F\_READ  
F\_READRAND  
F\_UNLOCK  
F\_WRITE  
F\_WRITERAND  
F\_WRITEZF

Table 2-14 lists error code definitions for register AL.

Table 2-14. BDOS Error Codes

Code	Definition
00H:	Function successful
01H:	Reading unwritten data
:	No available directory space (Write Sequential)
02H:	No available data block
03H:	Cannot close current extent
04H:	Seek to unwritten extent
05H:	No available directory space
06H:	Random record number out of range
* 08H:	Record locked by another process (restricted to files opened in Unlocked mode)
09H:	Invalid FCB (previous BDOS F_CLOSE system call returned an error code and invalidated the FCB)
0AH:	FCB checksum error
* 0BH:	Unlocked file unallocated block verify error
** 0CH:	Process record lock limit exceeded
** 0DH:	Invalid File ID
** 0EH:	No room in System Lock List
0FFH:	Physical error : refer to register AH
* - returned only for files opened in Unlocked Mode	
** - returned only by the F_LOCK and F_UNLOCK system calls for files opened in Unlocked mode	



For BDOS read and write system calls, the file system also sets register AH when the returned error code is a value other than zero or 0FFH. In this case, register AH contains the number of 128-byte records successfully read or written before the error was encountered. Note that register AH can only contain a nonzero value if the calling process has set the BDOS Multisector Count to a value other than one; otherwise register AH is always set to zero. On successful system calls (Error Code = 0), register AH is also set to zero. If the Error Code 0FFH, register AH contains a physical error code (see Table 2-15).

The following BDOS system calls return a directory code in register AL:

DRV\_SETLABEL  
F\_ATTRIB  
F\_CLOSE  
F\_DELETE  
F\_MAKE  
F\_OPEN  
F\_RENAME  
F\_SIZE  
F\_SFIRST  
F\_SNEXT  
F\_TIMEDATE  
F\_TRUNCATE  
F\_WRITEXFCB

The directory code definitions for register AL follow.

00H - 03H : successful function  
0FFH : unsuccessful function

With the exception of the F\_SFIRST and F\_SNEXT system calls, all functions in this category return with the directory code set to zero upon a successful return. However, for these two system calls, a successful directory code identifies the relative starting position of the directory entry in the calling process's current DMA buffer.

If a process uses the F\_ERRMODE system call to place the BDOS in Return Error mode, the following system calls return an error flag in register AL on physical errors:

DRV\_GETLABEL  
DRV\_ACCESS  
DRV\_SET  
DRV\_SPACE  
DRV\_FLUSH

The error flag definition for register AL follows.

00H : successful function  
0FFH : physical error : refer to register AH

The BDOS returns nonzero values in register AH to identify a physical or extended error if the BDOS Error mode is in one of the return modes. Except for system calls that return a Directory Code, register AL equal to 0FFH indicates that register AH identifies the physical or extended error. For functions that return a Directory Code, if register AL equals 255, and register AH is not equal to zero, register AH identifies the physical or extended error. Table 2-15 shows the physical and extended error codes returned in register AH.

Table 2-15. BDOS Physical and Extended Errors

<i>Code</i>	<i>Explanation</i>
01H	Disk I/O Error : permanent error
02H	Read/Only Disk
03H	Read/Only File, File Opened in Read/Only Mode, or File Password Protected in Write Mode and Correct Password Not Specified
04H	Invalid Drive : drive select error
05H	File Currently Open in an incompatible mode
06H	Close Checksum Error
07H	Password Error
08H	File Already Exists
09H	Illegal ? in FCB
0AH	Open File Limit Exceeded
0BH	No Room in System Lock List

The following two system calls represent a special case because they return an address in register AX.

DRV\_ALLOCVEC  
DRV\_DBP

When the calling process is in one of the BDOS return error modes and the BDOS detects a physical error for these system calls, it returns to the calling process with registers AX and BX set to 0FFFFH. Otherwise, they return no error code.

Under Concurrent CP/M-86, the following system calls also represent a special case.

DRV\_ALLRESET  
DRV\_RESET  
DRV\_SETRO

These system calls return to the calling process with registers AL and BL set to 0FFH if another process has an open file or has made a DRV\_ACCESS call that prevents the reset or write protect operation. If the calling process is not in Return Error mode, these system calls also display an error message identifying the process that prevented the requested operation.

*End of Section 2*

## Section 3

# Transient Commands

### 3.1 Transient Process Load and Exit

A transient process is a file of type CMD that is loaded from disk and resides in memory only during its operation. A resident system process is a file of type RSP that is included in Concurrent CP/M-86 during GENCCPM. Section 4 describes the three system memory models that determine the initial values of segment registers in transient processes.

You can initiate a transient process by entering a command at a system console. The console's TMP (Terminal Message Processor) then calls the Command Line Interpreter system call (refer to the P\_CLI system call), and passes to it the command line entered by the user. If the command is not an RSP, then the P\_CLI system call locates and then loads the proper CMD file. The P\_CLI system call calls the F\_PARSE system call that parses up to two filenames following the command, and places the properly formatted FCBs at locations 005CH and 006CH in the Base Page of the initial Data Segment.

The P\_CLI system call initializes memory, the Process Descriptor, and the User Data Area (UDA), and allocates a 96-byte stack area, independent of the program, to contain the process's initial stack. Concurrent CP/M-86 divides the DMA address into the DMA segment address and the DMA offset. The P\_CLI system call initializes the default DMA segment to the value of the initial data segment, and the default DMA offset to 0080H.

The P\_CLI system call creates the new process with a P\_CREATE system call and sets the initial stack so that the process can execute a Far Return instruction to terminate. A process also ends when it calls DRV\_ALLRESET or P\_TERM.

You can also terminate a process by typing a single CTRL-C during console input. See C\_MODE for details of enabling/disabling CTRL-C. CTRL-C also forces a DRV\_RESET call for each logged-in drive. This DRV\_RESET operation only affects removable media drives.

## 3.2 Command File Format

A CMD file consists of a 128-byte header record followed immediately by the memory image. The command file header record is composed of 8 group descriptors (GDs), each 9 bytes long. Each group descriptor describes a portion of the program to be loaded. The format of the header record is shown in Figure 3-1.

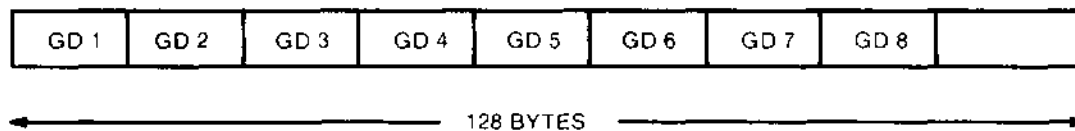


Figure 3-1. CMD File Header Format

In Figure 3-1, GD 1 through GD 8 represent group descriptors. Each group descriptor corresponds to an independently loaded program unit and has the format shown in Figure 3-2.

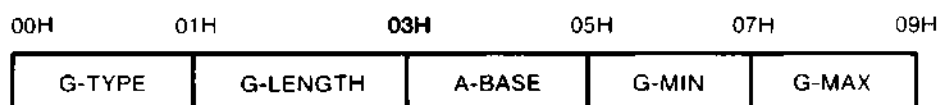


Figure 3-2. Group Descriptor Format

G\_Type determines the group descriptor type. The valid group descriptors have a G\_Type in the range 1 through 8, as shown in Table 3-1. All other values are reserved for future use. For a given CMD file header only a Code Group and one of any other type can be included.

If a program uses either the Small or Compact Model, the code group is typically pure; that is, it is not modified during program execution.

Table 3-1. Group Descriptors

<i>G_Type</i>	<i>Group Type</i>
01H	Code Group
02H	Data Group
03H	Extra Group
04H	Stack Group
05H	Auxiliary Group #1
06H	Auxiliary Group #2
07H	Auxiliary Group #3
08H	Auxiliary Group #4

All remaining values in the group descriptor are given in increments of 16-byte paragraph units with an assumed low-order 0 nibble to complete the 20-bit address.

Table 3-2. Group Descriptor Fields

<i>Field</i>	<i>Description</i>
G_Length	gives the number of paragraphs in the group. Given a G_length of 080H, for example, the size of the group is 0800H (2048 decimal) bytes.
A_Base	defines the base paragraph address for a nonrelocatable group.
G_Min/G_Max	define the minimum and maximum size of the memory area to allocate to the group.

The memory model described by a header record is implicitly determined by the group descriptors (refer to Section 4.1). The 8080 Model is assumed when only a code group is present, because no independent data group is named. The Small Model is assumed when both a code and data group are present but no additional group descriptors occur. Otherwise, the Compact Model is assumed when the CMD file is loaded.

### 3.3 Base Page Initialization

The Concurrent CP/M-86 Base Page contains default values and locations initialized by the P\_CLI and P\_LOAD system calls and used by the transient process.

The Base Page occupies the regions from offset 0000H through 00FFH relative to the initial data segment, and contains the values shown in Figure 3-3.



	L	M	H	L	H	
0	1	2	3	4	5	6
0	CODE LENGTH		CODE BASE		M80	
6	DATA LENGTH		DATA BASE		RESERVED	
C	EXTRA LENGTH		EXTRA BASE		RESERVED	
12	STACK LENGTH		STACK BASE		RESERVED	
18	AUX 1		AUX 1		RESERVED	
1E	AUX 2		AUX 2		RESERVED	
24	AUX 3		AUX 3		RESERVED	
2A	AUX 4		AUX 4		RESERVED	
30	BYTES 03H0 THROUGH 04FH ARE NOT CURRENTLY USED AND ARE RESERVED FOR FUTURE USE BY DIGITAL RESEARCH					
	.					
	.					
50	DRIVE	PASSWORD 1 ADDR		P1 LEN	PASSWORD 2 ADDR	
56	P2 LEN	RESERVED FOR FUTURE USE				
5C	DEFAULT FILE NAME1					
	.					
	.					
6C	DEFAULT FILE NAME2					
	.					
	.					
7C	CR	RANDOM RECORD NUMBER (OPT)				
80	DEFAULT 128-BYTE DMA BUFFER					

Figure 3-3. Concurrent CP/M-86 Base Page Values

The fields in the Base Page are defined as follows:

- The M80 byte is a flag indicating whether the 8080 Memory Model was used during load. The values of the flag are defined as:
  - 1 = 8080 Model
  - 0 = not 8080 ModelIf the 8080 Model is used, the code length never exceeds 0FFFFH.
- The bytes marked Aux 1 through Aux 4 correspond to a set of four optional independent groups that might be required for programs that execute using the Compact Memory Model. The initial values for these descriptors are derived from the header record in the memory image file.
- Length is stored using the Intel convention: low, middle, and high bytes.
- Base refers to the paragraph address of the beginning of the segment.
- The drive byte identifies the drive from which the transient program was read. 0 designates the default drive, while a value of 1 through 16 identifies drives A through P.
- Password 1 Addr (bytes 0051H-0052H) contains the address of the password field of the first command tail operand in the default DMA buffer at 0080H. The P\_CLI system call sets this field to 0 if no password is specified.
- P1 Len (byte 0053H) contains the length of the password field for the first command tail operand. The P\_CLI system call sets this to 0 if no password is specified.
- Password 2 Addr (bytes 0054H-0055H) contains the address of the password field of the second command tail operand in the default DMA buffer at 0080H. The P\_CLI system call sets this field to 0 if no password is specified.
- P2 Len (byte 0056H) contains the length of the password field for the second command tail operand. The P\_CLI system call sets this field to 0 if no password is specified.
- File Name1 (bytes 005CH-0067H) is initialized by the P\_CLI system call for a transient program from the first command tail operand of the command line.

- File Name2 (bytes 006CH-0077H) is initialized by the P\_CLI system call for a transient program from the second command tail operand of the command line.

**Note:** File Name1 can be used as part of a File Control Block (FCB) beginning at 05CH. To preserve File Name2, copy it to another location before using the FCB in file I/O system calls.

- The CR field (byte 007CH) contains the current record position used in sequential file operations with the FCB at 05CH.
- The optional Random Record Number (bytes 007DH-007FH) is an extension of the FCB at 05CH, used in random record processing.
- The Default DMA buffer (bytes 0080H-00FFH) contains the command tail when the P\_CLI system call loads a transient program.

### 3.4 Parent/Child Relationships

Under Concurrent CP/M-86, when one process creates another process, there is a parent/child relationship between them. The child process inherits most of the default values of the parent process. This includes the default disk, user number, console, list device, and password. The child process also inherits interrupt vectors 0, 1, 3, 4, 224, and 225, which the parent process initialized.

*End of Section 3*

.

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

## Section 4

# Command File Generation

### 4.1 Transient Execution Models

When the program is loaded, the initial values of the segment registers, the instruction pointer, and the stack pointer are determined by the specific type of memory model used by the transient process, indicated in the CMD file header record.

There are three memory models, the 8080 model, the Small Model, and the Compact Model, summarized in Table 4-1.

**Table 4-1. Concurrent CP/M-86 Memory Models**

<i>Model</i>	<i>Group Relationships</i>
8080 Model	Code and Data Groups Overlap
Small Model	Independent Code and Data Groups
Compact Model	Three or More Independent Groups

The 8080 Model supports programs that are directly translated from an 8080 environment where code and data are intermixed. The 8080 Model consists of one group that contains 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 can be addressed.

The Small 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.

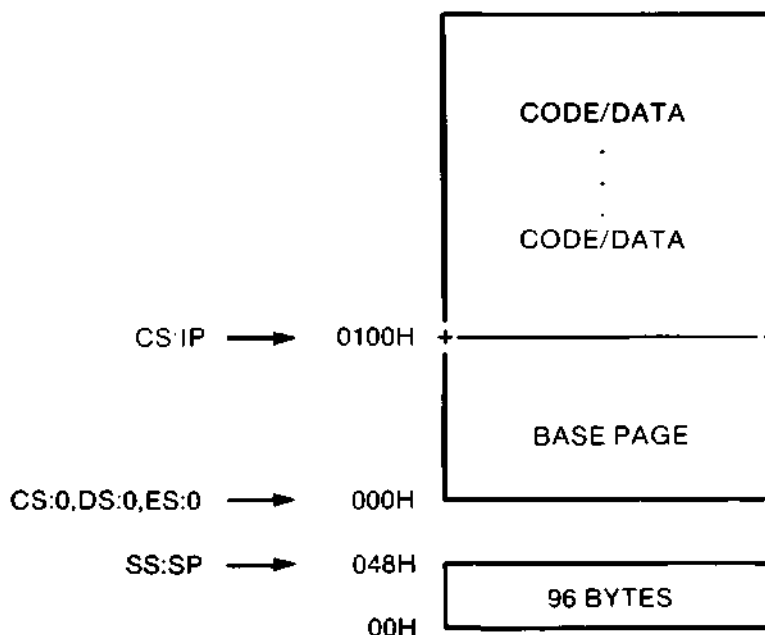
The Compact Model occurs when any of the extra, stack, or auxiliary groups are present in program. Each group can consist of one or more segments, but if any group exceeds one segment in size, or if auxiliary groups are present, then the application program must manage its own segment registers during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

These three models differ primarily in how the operating system initializes the segment registers when it loads a transient process. The P\_LOAD system call determines the memory model used by a transient program by examining the program group usage, as described in the following sections.

#### 4.1.1 The 8080 Memory Model

The 8080 Model is assumed when the transient program contains only a code group. In this case, the Command Line Interpreter (P\_CLI) system call initializes the CS, DS, and ES registers to the beginning of the code group and sets the SS and SP registers to a 96-byte initial stack area that it allocates.

**Note:** the P\_CLI system call initializes the stack so that if the process executes a Far Return instruction, it terminates. This system call sets the Instruction Pointer (IP) Register to 100H, thus allowing Base Page values at the beginning of the code group. Following program load, the 8080 Model appears as shown in Figure 4-1.



**Figure 4-1. Concurrent CP/M-86 8080 Memory Model**

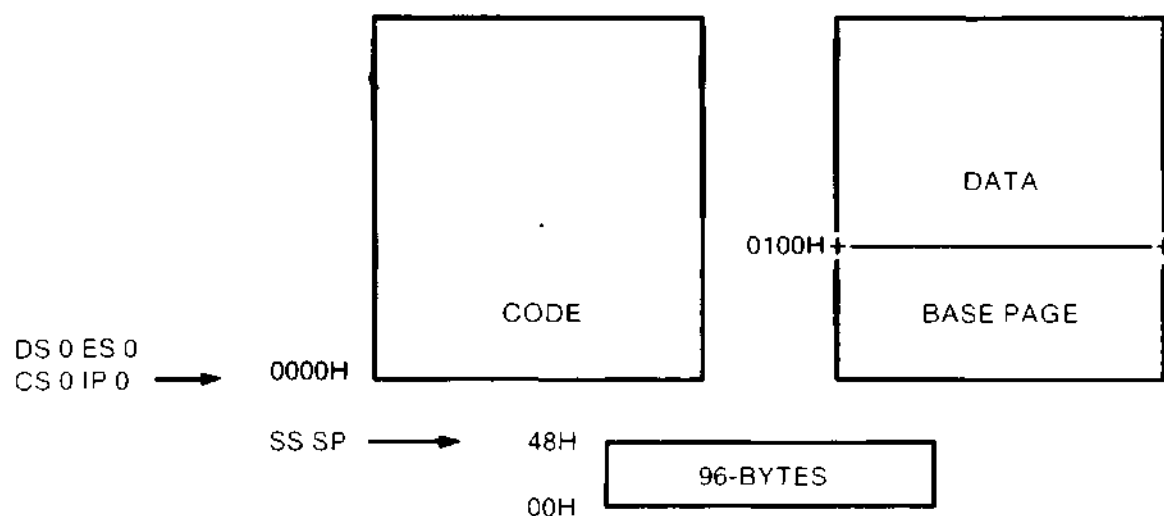
The intermixed code and data areas are indistinguishable. The Base Page values are described in Section 3.3. The following ASM-86 example shows how to code an 8080 Model transient assembly language program.

```
        cseg
        org    100h
        .
        .      (code)
endcs   equ    $
        dseg
        org    offset endcs
        .
        .      (data)
        end
```

#### 4.1.2 The Small Memory Model

The Small Model is assumed when the transient program contains both a code and data group. (In ASM-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 model, the P\_CLI system call sets the CS register to the beginning of the data group, and the SS and SP registers to a 96-byte initial stack area that it initializes. Following program load, the Small Model appears as shown in Figure 4-2.





**Figure 4-2. Concurrent CP/M-86 Small Memory Model**

The machine code begins at CS+0000H, the Base Page values begin at DS+0000H, and the data area starts at DS+0100H. The following ASM-86 example shows how to code a Small Model transient assembly language program.

```
cseg
.
.      (code)
dseg
org    100h
.
.      (data)
end
```

### 4.1.3 The Compact Memory Model

The Compact Model is assumed when code and data groups are present, along with one or more of the remaining stack, extra, or auxiliary groups. In this case, the P\_CLI system call 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 a 96-byte stack area allocated by this system call.

Figure 4-3 shows the initial configuration of the segments in the Compact Model. The values of the various segment registers can be changed during execution by loading from the initial values placed in Base Page. This allows access to the entire memory space.

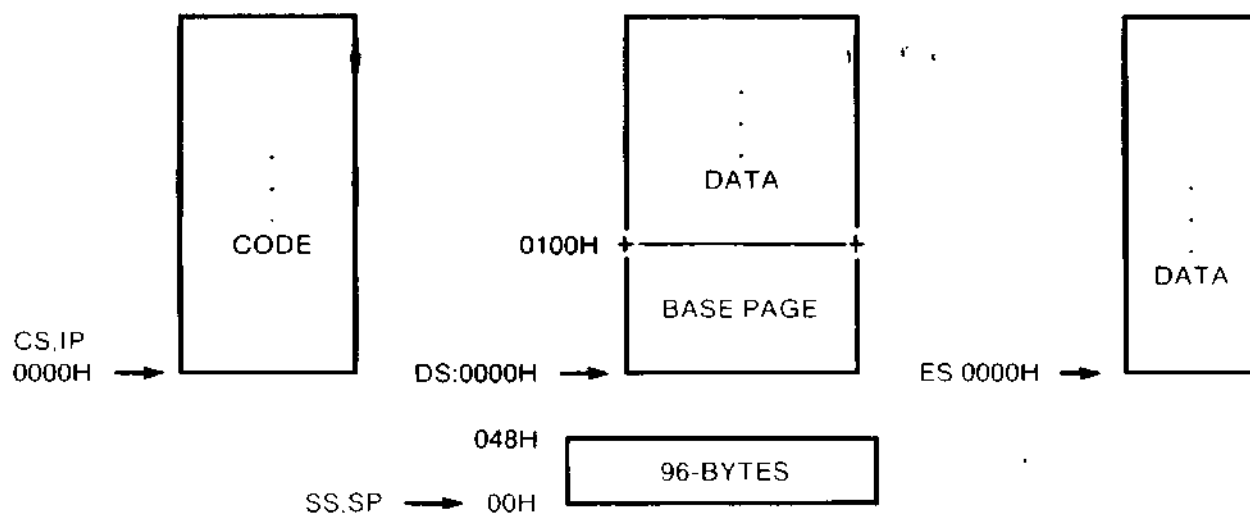


Figure 4-3. Concurrent CP/M-86 Compact Memory Model

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 P\_CLI system call 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.

The following ASM-86 example shows how to code a Compact Model assembly language transient program.

```
cseg
.
.      (code)
dseg
org    100h
.
.      (data)
eseg
.
.      (more data)
sseg
.
.      (stack area)
end
```

## 4.2 GENCMD

The GENCMD utility creates a CMD file from an input HEX file. GENCMD does not alter the original HEX file. The GENCMD invocation has the following form:

GENCMD filename {parameter-list}

where the filename corresponds to the HEX input file with an assumed and unspecified filetype of H86. GENCMD accepts optional parameters to specifically identify the 8080 Model and to describe memory requirements of each segment group. The GENCMD parameters are listed following the filename, as shown in the command line above where the parameter list consists of a sequence of keywords (shown below) and values separated by commas or blanks.

8080 CODE DATA EXTRA STACK X1 X2 X3 X4

The 8080 keyword forces a single code group so that the P\_LOAD system call sets up the 8080 Model for execution, allowing intermixed code and data in a single segment. The form of this command is

GENCMD filename 8080

The remaining keywords follow the filename or the 8080 option and define specific memory requirements for each segment group, corresponding one-to-one with the segment groups defined in the previous section. In each case, the values corresponding to each group are enclosed in square brackets and separated by commas. Each value is a hexadecimal number representing a paragraph address or segment length in paragraph units denoted by hhhh, prefixed by a single letter that defines each value:

Ahhhh	Load the group at absolute location hhhh
Bhhhh	The group starts at hhhh in the hex file
Mhhhh	The group requires a minimum of hhhh * 16 bytes
Xhhhh	The group can address a maximum of hhhh * 16 bytes

Generally, the CMD file header record values are derived directly from the HEX file and the parameters shown above need not be included. The following situations, however, require the use of GENCMD parameters.

- The 8080 keyword is included whenever ASM-86 is used in the conversion of 8080 programs to the 8086/8088 environment when code and data are intermixed within a single 64K segment, regardless of the use of CSEG and DSEG directives in the source program.
- An absolute address (a hexadecimal value) must be given for any group that must be located at an absolute location. This value is not usually specified, as Concurrent CP/M-86 cannot ensure that the required memory region is available. In that case the CMD file cannot be loaded.
- The B value is used when GENCMD processes a HEX file produced by Intel's OH86 or a similar utility program that contains more than one group. The output from OH86 consists of a sequence of data records with no information to identify code, data, extra, stack, or auxiliary groups. In this case, the B value marks the beginning address of the group named by the keyword, causing GENCMD to load data following this address to the named group (refer to the examples below). Thus, the B value is usually used to mark the boundary between Code and Data Segments when no segment information is included in the HEX file. Files produced by ASM-86 do not require the use of the B value because segment information is included in the HEX file.
- The minimum memory value (M value) is included only when the HEX records do not define the minimum memory requirements for the named group. Generally, the code group size is determined precisely by the data records loaded into the area. The total space required for the group is defined by the range between the lowest and highest data byte addresses. The data group, however, might contain uninitialized storage at the end of the group. Thus no data records are present in the HEX file that define the highest referenced data item. The highest address in the data group can be defined within the source program by including the ASM86 directive DB 0 as the last data item in the assembly language source file. Alternatively, the M value can be included to allocate the additional space at the end of the group. Similarly, the stack, extra, and auxiliary group sizes must be defined using the M value unless the highest addresses within the groups are implicitly defined by data records in the HEX file.

- The maximum memory size, given by the X value, is generally used when additional free memory might be needed for such purposes as I/O buffers or symbol tables. If the data area size is fixed, then the X parameter need not be included. In this case, the X value is assumed to be the same as the M value. The value XFFFF allocates the largest memory region available but, if used, the assembly language transient program must be aware that a three-byte length field is produced in the Base Page for this group where the high-order byte might be nonzero. Programs converted directly from an 8080 environment or programs that use a 2-byte pointer to address buffers should restrict this value to XFFF or less, producing a maximum allocation length of 0FFF0H bytes.

The following GENCMD command line transforms the file X.H86 into the file X.CMD with the proper header record:

```
0A>GENCMD x code[a40] data[m30,xfff]
```

In this case, the code group is forced to paragraph address 40H or its equivalent, byte address 400H. The data group requires a minimum of 300H bytes, but can use up to 0FFF0H bytes, if available.

Assuming a file Y.H86 exists on drive B containing Intel HEX records with no interspersed segment information, the command

```
0A>GENCMD b : y data[b30,m20] extra [b50] stack [m40]x1[m40]
```

produces the file Y.CMD on drive B by selecting records beginning at address 0000H and less than 0300H for the Code Segment, with records starting at 0300H and less than 0500H allocated to the Data Segment. The Extra Segment is filled from records beginning at 0500H and higher, while the Stack and Auxiliary Segment #1 are uninitialized areas requiring a minimum of 0400H bytes each. In this example, the data area requires a minimum of 0200H bytes. Note again that the B value need not be included if the Digital Research ASM-86 assembler is used.

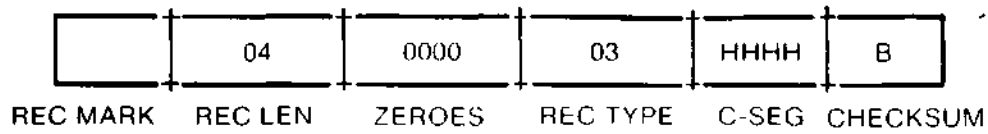
## 4.3 Intel Hexadecimal File Format

GENCMD input must be in Intel hexadecimal file format, produced by both the Digital Research ASM-86 assembler and the standard Intel OH86 utility program. (Refer to *Intel MCS-86 Software Development Utilities Operating Instructions for ISIS-II™ Users*, published by Intel.) The CMD file produced by GENCMD contains a header record defining the memory model and memory size requirements for loading and executing the CMD file.

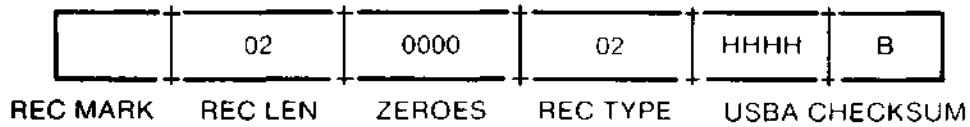
An Intel hexadecimal file consists of the traditional sequence of ASCII records where the beginning of the record is marked by an ASCII colon, and each subsequent digit position contains an ASCII hexadecimal digit in the range 0-9 or A-F.

There are four kinds of hexadecimal record formats. The Start Address Record specifies the starting address of the execution file. The Extended Address Record specifies the bits 4-19 of the Segment Base Address, where bits 0-3 of the SBA are zero. The Data Record contains a string of hexadecimal ASCII code that represents a portion of the 8086 memory image. The End-of-File record specifies the end of the object file.

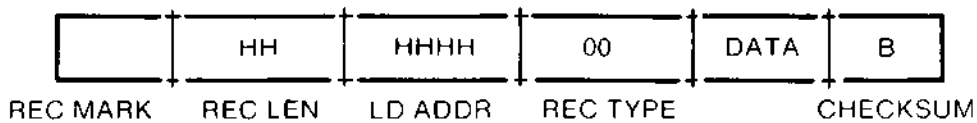
Figure 4-4 shows the four record formats, their fields, and the contents of these fields. The fields are defined in Table 4-2.



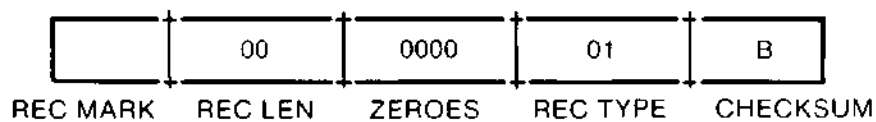
STARTING ADDRESS RECORD



EXTENDED ADDRESS RECORD



DATA RECORD



END OF FILE RECORD

**Figure 4-4. Intel Hexadecimal File Formats**



Table 4-2. Intel Hex Field Definitions

<i>Field</i>	<i>Contents</i>
Rec Mark	Specifies start of record
Rec Len	Record Length 00-FF (0-255 in decimal)
Zeroes	Extended Address Record: 0000H Starting Address Record: 0000H End-of-File Record: 0000H
Ld Addr	Data Record: SBA offset defining address of byte 0 of data
Rec Type	00 = Data Record 01 = End of File Record 02 = Extended Address Record 03 = Starting Address Record  The following are output from ASM-86 only:  81 same as 00, data belongs to Code Segment 82 same as 00, data belongs to Data Segment 83 same as 00, data belongs to Stack Segment 84 same as 00, data belongs to Extra Segment * 85 paragraph address for absolute Code Segment * 86 paragraph address for absolute Data Segment * 87 paragraph address for absolute Stack Segment * 88 paragraph address for absolute Extra Segment
C-Seg	Four hexadecimal digits specifying the Code Segment address. The high-order and low-order digits are the 10th and 13th characters of the record, respectively.

Table 4-2. (continued)

<i>Field</i>	<i>Contents</i>
USBA	Four hexadecimal digits specifying the Upper Segment Base Address. The high-order and low-order digits are the 10th and 13th characters of the record, respectively.
data	Pairs of hexadecimal digits representing the ASCII code for each data byte. The high-order digit is the first digit of each pair.
Checksum	<p>Extended Address Record: Checksum of Rec Len, Zeroes, Rec Type, and USBA fields.</p> <p>Starting Address Record: Checksum of Rec Len, Zeroes, Rec Type, C-Seg, and IP fields.</p> <p>Data Record: Checksum of Rec Len, Ld Addr, Rec Type, and data fields.</p> <p>End-of-File Record: Contains ASCII code 4646H, checksum of Rec Len, Zeroes, and Rec Type fields.</p>
* 85, 86, 87, and 88 are Digital Research Extensions.	

All characters preceding the colon for each record are ignored. See *MCS -86 Absolute Object File Formats*, published by Intel, for additional information on hexadecimal file record format.

*End of Section 4*

# Section 5

## Resident System Process Generation

### 5.1 Introduction to RSPs

428 10/01/88 1.5.7

Resident System Processes are programs that become part of the Concurrent CP/M-86 operating system. They can be useful in several ways: to create a turnkey system, autoloading programs when Concurrent CP/M-86 is booted; to build customized user interfaces or shells at the consoles, for monitoring hardware not supported in the XIOS; and to avoid disk loading time for frequently used commands.

The source code for the ECHO RSP is included in Appendix F. Study this listing carefully while reading this section. The discussion of the P\_CREATE system call in Section 6 is also helpful in understanding RSPs.

Resident System Processes are included in Concurrent CP/M-86 during system generation. GENCCPM searches the directory for all files with the filetype RSP and prompts the user to choose whether it is to be included in the generated system file, CCPM.SYS. An RSP file is created by generating a CMD file and renaming it with an RSP filetype. The GENCCPM program is documented in the *Concurrent CP/M-86 System Guide*.

### 5.2 RSP Memory Models

Under Concurrent CP/M-86, there are two basic memory models for RSPs. They are similar to the 8080 Model and the Small Model of transient programs. However, several important distinctions exist between the transient program and RSP memory models. The RSP has no equivalent to the Base Page of the transient program's Data Segment. The RSP is responsible for its own Process Descriptor (PD) and User Data Area (UDA). The system creates and initializes these data structures for the transient programs automatically at load time. RSPs, on the other hand, must initialize these structures within their own Data Segments.

Note that Concurrent CP/M-86 does not support compact model RSPs. Extra and Stack Segments must be part of the Data Segment.

Although there is no Base Page in an RSP, there is an RSP header that must exist at offset 00H of the Data Segment. In the 8080 Model, this implies that the RSP header is in the Code Segment. The RSP header and the associated data structures are discussed in Section 5.4.

### 5.2.1 8080 Model RSP

The 8080 Model consists of mixed code and data. When the system gives control of the CPU to an 8080 Model RSP, it initializes the Code, Data, Extra and Stack Segment registers to the same value. Use GENCMD with the 8080 option to generate an 8080 Model RSP. GENCCPM assumes the 8080 Model if the CMD File Header Record of the RSP has a single Code Group Descriptor and no other Group Descriptors (refer to Section 3.2). When discussing an 8080 Model RSP, any reference to the Data Segment also refers to the Code Segment.

### 5.2.2 Small Model RSP

The Small Model RSP implies separate Code and Data Segments. Before the system gives control of the CPU to a Small Model RSP, it initializes the Data, Extra and Stack Segment Registers to the Data Segment address, while the Code Segment register is initialized to the Code Segment address. There is no guarantee where GENCCPM will place the Code Segment in memory relative to the Data Segment. The CMD Header Record for this kind of RSP must have both Data and Code Group Descriptors.

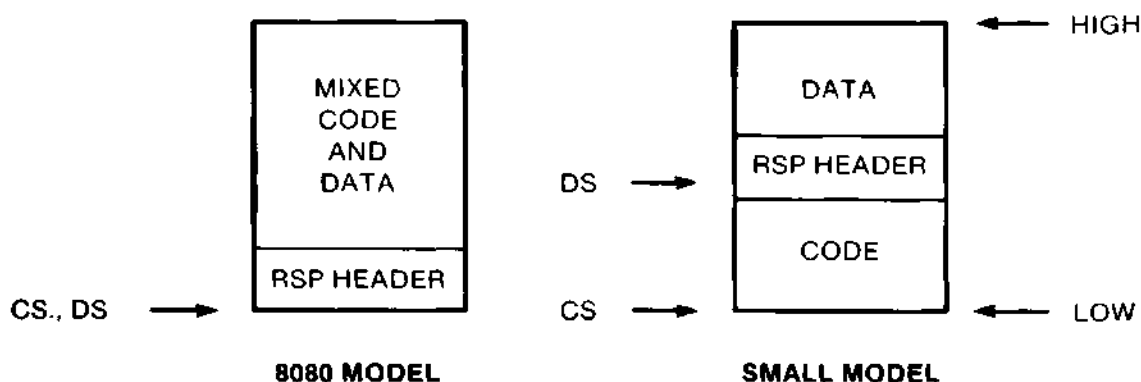
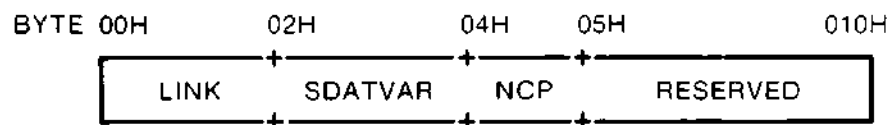


Figure 5-1. 8080 and Small RSP Models

### 5.3 Multiple Copies of RSPs

At system generation, GENCCPM can make up to 255 extra copies of an RSP, such that each copy generates a separate process running under Concurrent CP/M-86. GENCCPM accomplishes this by making multiple copies of the RSP; and initializing each to be a separate RSP. The number of copies made by GENCCPM can be fixed, or dependent on a byte value in the System Data Area. To determine the number of copies to make, GENCCPM looks at two fields in the RSP Header. The format of the RSP Header is shown in figure 5-2.



**Figure 5-2. RSP Header Format**

If the SDATVAR field is nonzero, it is used as an offset of a byte value in the System Data Area, which contains the number of copies to be generated. The offset should indicate a value that is set by the user during GENCCPM. The TMP RSP uses this feature by placing the offset of the NVCNS (Number of Virtual Consoles) field into the SDATVAR field. This way, a TMP is generated for each System Console specified by the user. If SDATVAR is 0 then the NCP byte in the RSP header is used as the number of extra copies to make. If both of these fields in the RSP Header are 0 then no extra copies are made, and only a single RSP is created. The ECHO RSP is an example of the latter.

If the number of extra copies is determined by GENCCPM to be greater than 0, each copy of the RSP is given a unique copy number. The copy number is placed in the NCP field and the ASCII equivalent is appended to the end of the Process Descriptor NAME field of each copy. If there is not enough space for the number in the PD NAME, part of the PD NAME is over written. For the example TMP RSP, GENCCPM makes the specified number of copies and changes the NAME field in each copy to be TMP0, TMP1, TMP2, ..., and sets the NCP field to 0, 1, 2, ..., respectively.

### 5.3.1 8080 Model

When GENCCPM makes copies of an 8080 Model RSP, the CS, DS, ES, and SS fields in each copy's User Data Area are set to the paragraph address where the RSP is in memory after loading.

### 5.3.2 Small Model

If multiple copies of a Small Model RSP are to be generated, GENCCPM copies both the Code and Data Groups of the RSP, if the MEM field of the Process Descriptor is 0. See the P\_CREATE system call for a description of the Process Descriptor format. GENCCPM sets the UDA fields CS to the Code Segment of the RSP and DS, ES and SS to the Data Segment of the RSP.

### 5.3.3 Small Model with Shared Code

If a Small Model RSP has a nonzero MEM field in its Process Descriptor, the Code Segment is assumed to be reentrant. When copies are made of this type of RSP only the Data Group is copied. GENCCPM sets the UDA CS field for each copy to the paragraph address of the one Code Segment for the RSP's. The DS, ES, and SS, in each copied Data Segment, are set by GENCCPM to the paragraph address of the Data Segment for that particular copy.

## 5.4 Creating and Initializing an RSP

An RSP that is to be invoked from a console, or through the P\_CLI system call, must create a special queue called an RSP Command Queue. Such an RSP is called a Command RSP. This type of RSP usually performs some initialization routine and then goes into a loop. The initialization routine consists of creating and opening an RSP Command Queue as well as changing the priority to the default transient process priority. (Priority values with regard to RSPs follow.)

The first step of the loop reads a message from the RSP Command Queue. The process that writes the message to the RSP Command Queue activates the associated RSP. After the RSP returns from the `Q_READ` system call, it obtains the system resources it needs, such as the calling process's console. Typically, the RSP process is assigned the console process by the CLI after the CLI has succeeded in writing the command tail to the RSP Queue. This is only true if the RSP Process Descriptor name matches the RSP Command Queue name. Refer to the `P_CLI` (Call Command Line Interpreter) system call description for information about how the CLI handles a command.

When the RSP completes its activities for the given command, it releases any system resources it has acquired, including the console, and restarts the loop by reading from its RSP Command Queue. A Command RSP is a single process and is a serially reusable resource; in other words, the RSP acts on one message at a time. When several processes attempt to invoke a single Command RSP, they wait as described in the `Q_READ` and `Q_CREAD` system call in Section 6. Refer to these and to the `Q_WRITE` and `Q_CWRITE` system calls for further details.

**Note:** it is certainly possible to create RSPs that are invoked differently.

The format of the RSP Command Queue Message is shown in Figure 5-3.

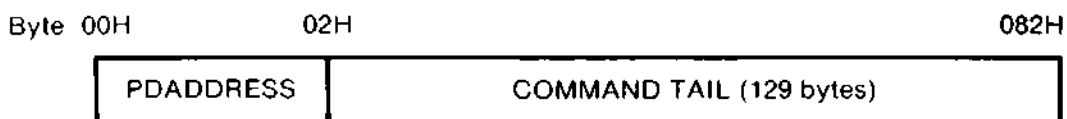


Figure 5-3. RSP Command Queue Message

The PDADDRESS is the offset relative to the System Data Area segment of the Process Descriptor of the process calling the RSP. A program that wants to invoke an RSP and is forming an RSP Command Queue Message, can find its Process Descriptor address by calling the P\_PDADR system call. The COMMAND TAIL usually contains what the TMP sends to the CLI minus the command name, and is terminated with a zero byte.

When a command is entered at a console, the TMP performs a P\_CLI system call. The P\_CLI system call attempts to open a queue that has the RSP Flag on and has the same name as the command sent to the CLI. If the Q\_OPEN is successful, the P\_CLI system call attempts to assign the calling process's console to a process with the same name as the command. The P\_CLI system call then creates an RSP Command Queue Message with the command tail sent to the CLI from the TMP, and writes it to the RSP Command Queue (refer to the discussion of the P\_CLI and Q\_WRITE system calls in Section 6). A transient program can use a Command RSP in the same manner by writing directly to the appropriate RSP Command Queue. An advantage of using the P\_CLI system call is that it looks for an RSP first and only searches on disk for a CMD file if the RSP is not found.

When an RSP reads an RSP Command Queue Message, it often needs information about the calling process, such as which console, list device, drive, or user number to use. If an RSP is invoked through the P\_CLI system call, the RSP is assigned the calling process's console, but if the RSP Command Queue is written to directly, the calling process might or might not assign its console to the RSP. A Command RSP can use the PD address in the Command RSP Message to find out what the default devices of the calling process are. The RSP should release any resources it assigns to itself when it is finished.



The beginning of the RSP Data Segment has a fixed format starting at offset 0. This data structure is the RSP Header. Note that in the 8080 Model, the RSP Header is also in the Code Segment. After the RSP Header is a Process Descriptor starting at offset 010H. A User Data Area and a stack must also be within the Data Segment, with the UDA placed at a paragraph boundary relative to the beginning of the Data Segment. If system calls assuming a default DMA buffer are used, a 128-byte DMA Buffer must also exist. The DMA OFFSET field in the User Data Area should be set to the address of the DMA buffer. When the process is created by Concurrent CP/M-86, the DMA SEGMENT field is initialized to the same value as the DS register. The DMA SEGMENT and OFFSET can also be set by calling F\_DMASEG and F\_DMAOFF once the RSP is running. The beginning of the RSP Data Segment is shown in Figure 5-4.

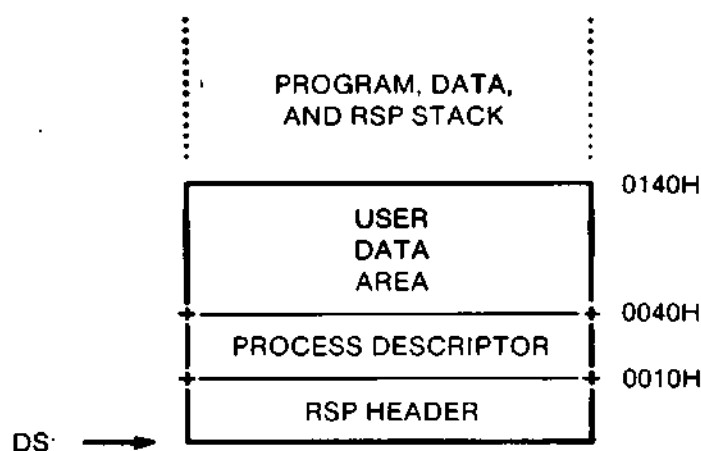


Figure 5-4. RSP Data Segment

The RSP Header must be located at offset zero in the RSP Data Segment, the RSP Process Descriptor must be at offset 010H, and the RSP User Data Area must begin on an even paragraph boundary.

#### 5.4.1 The RSP Header

As discussed in Section 5.2, the number of copies made of an RSP is dependent on the values of the SDATVAR and NCP fields in the RSP Header. If no copies are desired, these fields must be zero. As a convenience, when Concurrent CP/M-86 creates the RSP process, the LINK field in the RSP Header is set to the paragraph address of the System Data Area. The System Data Area can always be obtained by an RSP or transient program with the S\_SYSDAT system call.

#### 5.4.2 The RSP Process Descriptor

The RSP Process Descriptor should be initialed to zeros, except for the PRIORITY, FLAGS, NAME, and UDA SEGMENT fields. The PRIORITY field is usually initialized to 190. This is higher than transient programs and TMPs (200 and 198 respectively), but lower than the INIT process, which has a priority of 1. The description of the P\_PRIORITY system call in Section 6 contains more information about system priority assignments.

Starting an RSP at a priority of 190 ensures that the RSP is able to create and open an RSP Command Queue before it can be invoked through a TMP. RSPs such as ECHO usually set their priority to 200 after creating and opening their RSP Command queue and before attempting to read from the queue.

**Note:** there are no guarantees about the order in which the RSP processes are created by the Concurrent CP/M-86 operating system. If one RSP must run before another, it must have a higher priority. Such is the case when one RSP uses a resource created by a second RSP; the second must run (at least during initialization) with a priority higher than the first.

The Process Descriptor SYS and KEEP Flags can be initialized in the RSP Data Segment (refer to P\_CREATE in Section 6 for further flag details). The SYS Flag allows a process to read and write to and from restricted system queues. This is discussed below with regard to RSP Command Queues. The KEEP flag signals to the operating system that this process cannot be terminated. This flag is necessary if an RSP is not to be terminated when a CTRL-C is typed on a console being used by the RSP.

The NAME field of the RSP's Process Descriptor is 8 bytes long. It is assumed to be left-justified and padded with blanks on the right. If an RSP Command Queue is going to be used to invoke the RSP through the CLI, the PD must have the same upper-case name as the Command Queue. The UDA field in the Process Descriptor must be the offset in paragraphs of the UDA relative to the RSP data segment. GENCCPM restores the UDA field in the Process Descriptor to the actual UDA paragraph address when the system is loaded.

If the PD field name is not the same as the Command Queue, the console is not assigned to the RSP by the CLI.

#### 5.4.3 The RSP User Data Area

The User Data Area must have its SP field set to the offset of a three-word IRET structure, in the RSP's Data Segment. The offset is relative to the beginning of the Data Segment. The first of the three words is the offset of the code entry point for the RSP, relative to the beginning of the RSP Code Segment. Concurrent CP/M-86 executes an IRET instruction to start the RSP using these three words for the IP, CS and Flag registers respectively. The CS value on the stack is initialized to be the CS field of the UDA, while the Flag value is set to 0200H (interrupts on). The RSP stack must come immediately before these three words.

The initial values of the AX, BX, CX, DX, DI, SI, and BP registers are taken from the appropriate fields in the UDA.

The DMA OFFSET field should be set to the offset of the DMA buffer in the RSP's Data Segment. Except for the SP and DMA OFFSET fields, and possibly the AX, BX, CX, DX, DI, SI, and BP fields, the remainder of the UDA fields should be initialized to 0. The CS, DS, ES, and SS fields are set by GENCCPM as discussed in Section 5.3.

#### 5.4.4 The RSP Stack

The RSP must reserve space for its stack, which is assumed to lie within the RSP's Data Segment. This stack must be large enough to accommodate what the RSP code will need, plus four levels (eight bytes) to handle possible hardware interrupts. We highly recommend that you reserve more than four extra levels of stack.

The SP field in the RSP's UDA points to the top of this stack; the top contains the three-word IRET instruction discussed above.

#### 5.4.5 The RSP Command Queue

The RSP's Command Queue contains information that determines when it begins execution, and which console it is attached to. If an RSP is to be accessible from a console via the TMP, the Command Queue name must be in upper-case. The FLAGS field in the RSP Command Queue Descriptor must have the RSP bit on. If this flag is not on, the CLI will not write a message to the RSP Command Queue, and instead attempts to load a transient program. The KEEP flag should be set on to protect the RSP QUEUE from inadvertent use of the Q\_DELETE system call.

The RESTRICTED flag (refer to the Q\_MAKE system call in Section 6) makes a QUEUE accessible only by privileged processes. Privileged processes have the SYS Flag on in their Process Descriptor. If the RESTRICTED Flag is on in an RSP Command Queue, then only privileged processes can invoke the related RSP. A lower-case letter in the RSP Command Queue name and the RESTRICTED Flag provide two methods of filtering access to an RSP QUEUE.

The Queue Descriptor of the RSP Command Queue must have a message length of 131 bytes. The format of this message is shown above. The number of messages is usually 1. If the Queue Descriptor is within 64K bytes of the beginning of the System Data Area, buffer space for the Queue Descriptor must be allocated in the RSP. The BUFFER field in the Queue Descriptor must be the offset of this buffer, relative to the beginning of the RSP's Data Segment. The buffer size is the message length times the number of messages, usually 131 bytes.

**Note:** the queue buffer should be before the Queue Descriptor within the RSP Data Segment.

An RSP can certainly create other queues besides the RSP Command Queue used with Command RSPs. However, any queue an RSP creates that lies within 64K of the System Data Area must have a buffer area pointed to by the BUFFER field in its Queue Descriptor. To be safe, the buffer should come before the Queue Descriptor in the RSP's Data Segment. It is assumed the BUFFER field points to a buffer that is also within 64K of the System Data Area. If the Queue Descriptor is farther than 64K from the System Data Area, Concurrent CP/M-86 uses buffer space in the System Data Area. Refer to the Q\_MAKE system call in Section 6 for further details.

In order to open the RSP Command Queue and subsequently read from it, a Queue Parameter Block and its associated buffer must be allocated in the RSP's Data Segment. These structures are treated just as in a transient process. For any queues created by an RSP, it is stressed that the queue buffer areas associated with the Queue Descriptor and the Queue Parameter Block are separate, distinct areas of storage.

#### 5.4.6 Multiple Processes within an RSP

An RSP can create child processes by calling the P\_CREATE system call. Note that if the Process Descriptor of the process being created is within 64K bytes of the beginning of the System Data Area, the PD structure is used directly by Concurrent CP/M-86. Otherwise the PD structure is copied into the PD table in the System Data Area.

2 months to 3000

## 5.5 Developing and Debugging an RSP

The first RSP you attempt should be very simple, on the order of complexity of the ECHO RSP listed in Appendix D. New RSPs should be developed and debugged as if they were transient processes, such as Concurrent CP/M-86 CMD utilities, then converted into RSPs.

An RSP debugging session should proceed like an XIOS debugging session: first load CP/M-86, then invoke DDT-86™, and then bring up Concurrent CP/M-86. The *Concurrent CP/M-86 System Guide* provides more information about running Concurrent CP/M-86 under CP/M-86.

After reading in the CCPM.SYS file under DDT-86, find the RSPSEG field of the System Data Segment (SYSDAT). The paragraph address of the SYSDAT is found in the A\_BASE field of the Data Group Descriptor in the CCPM.SYS command file header. The CMD header is described in Section 3.2 and the Sysdat area is described in the S\_SYSDAT system call in Section 6. The RSPSEG field contains the paragraph address of the Data Segment of the first RSP in a linked list of the RSPs included by GENCCPM.

By using the Display Memory (D) command of DDT-86 to show memory at the segment RSPSEG, the name of the first RSP can be identified in the RSP's Process Descriptor. The LINK field in the RSP Header, which will be the first word in the RSPSEG segment, is the paragraph value of the next RSP's Data Segment. A zero in the LINK field means the end of the list of RSPs. Note that linkage information is lost once Concurrent CP/M-86 is initialized. The LINK field of the RSP Header contains the System Data Segment once an RSP begins execution.

Once the RSP to be debugged is located, the initial code entry point can also be found. As discussed previously, the SP field in the RSP's UDA is the offset from the beginning of the RSP's Data Segment of the three-word IRET structure. The first word of the IRET structure contains the initial value of the IP register when Concurrent CP/M-86 creates the RSP process. The initial value of the CS register is in the CS field also in the RSP's UDA. Once this is done, you can set break points in the RSP, similar to setting break points in XIOS system calls.

*End of Section 5*

## Section 6

# System Calls

This section describes the Concurrent CP/M-86 system calls in tabular form. It is intended both as an introduction to the calls and as a reference for use during programming. You should be familiar with the material in Sections 1 through 5 before proceeding.

The first table, Table 6-1, describes the categories of Concurrent CP/M-86 system calls and their general uses. Table 6-2 summarizes the Concurrent CP/M-86 system calls. Use it as a quick reference to find the system call you need while programming. The system calls are broken down into functional groups. Immediately following is Table 6-3, a cross-reference showing the system calls in numerical order. Table 6-4 is an index providing the page numbers and figure titles of commonly used data structures. Table 6-5 lists the error codes returned in register CX.

Table 6-1. System Call Categories

<i>Category</i>	<i>Use</i>
<b>C_ Console System Calls</b>	The Console System Calls handle I/O operations for virtual consoles on a character, string, and line basis, attach and detach consoles from processes, and return or change the number corresponding to the default virtual console.
<b>DEV_ Device System Calls</b>	The Device System Calls deal with flags and polling in managing system resources.
<b>DRV_ Disk Drive System Calls</b>	The Disk Drive System Calls manage Concurrent CP/M-86 logical drives.
<b>F_ File-Access System Calls</b>	The File-Access System Calls include calls that operate on files within a directory, calls that operate on records within files, and other miscellaneous system calls related to file I/O.
<b>L_ List Device System Calls</b>	The List Device System Calls write characters or strings to the default list device, attach and detach the default list device from calling processes, and return or change the number corresponding to the default list device.
<b>M_ MP/M-86.. Memory Management System Calls</b>	The M_ Memory Management System Calls are included for compatibility with MP/M-86. These calls allocate and free memory segments according to the MP/M-86 segmentation algorithm.



Table 6-1. (continued)

<i>Category</i>	<i>Use</i>
MC_ CP/M-86 Memory Management System Calls	The MC_ Memory Management System Calls allocate and free memory segments according to the CP/M-86 segmentation algorithm.
P_ Process/Program System Calls	The Process/Program System Calls create and terminate processes, call other processes, and perform other operations on processes.
Q_ Queue Management System Calls	The Queue Management System Calls create, delete, open, read from, and write to queues.
S_ System Calls	The System Calls return various types of systems data, such as version numbers and addresses.
T_ Time System Calls	The Time System Calls set the system calendar and clock and return the time from them in hours and minutes or in hours, minutes, and seconds.

Table 6-2. Concurrent CP/M-86 System Calls

Number Dec Hex		Mnemonic	Definition
Console I/O System Calls			
149	95	C_ASSIGN	Assign default virtual console to another process.
146	92	C_ATTACH	Establish ownership of the default virtual console to the calling process; suspend process until console becomes available.
162	A2	C_CATTACH	Conditionally establish ownership of the default virtual console by the calling process; return an error message if the device is unavailable.
110	6E	C_DELIMIT	Set or return current String Output Delimiter. Used with C_WRITESTR.
147	93	C_DETACH	Detach default virtual console from the calling process.
153	99	C_GET	Return the virtual console number of the calling process.
109	6D	C_MODE	Set or return Console mode.
6	06	C_RAWIO	Perform Raw mode I/O with the default virtual console.
1	01	C_READ	Read a character from the default virtual console.
10	0A	C_READSTR	Read an edited line from the default virtual console.

Table 6-2. (continued)

<i>Number</i> <i>Dec Hex</i>		<i>Mnemonic</i>	<i>Definition</i>
148	94	C_SET	Set or change the default virtual console for the calling process.
11	0B	C_STAT	Obtain the input status of the default virtual console.
2	02	C_WRITE	Write a character to the default virtual console.
111	6F	C_WRITEBLK	Write a specified number (block) of characters to the default virtual console.
9	09	C_WRITESTR	Write a string to the default virtual console until delimiter.
Device System Calls			
131	83	DEV_POLL	Poll a noninterrupt-driven device.
133	85	DEV_SETFLAG	Set a system flag.
132	84	DEV_WAITFLAG	Wait for a system flag to be set before restoring the current process.
Disk Drive System Calls			
38	26	DRV_ACCESS	Indicate access to specified drives.
27	1B	DRV_ALLOCVEC	Get the address of the disk Allocation Vector.

Table 6-2. (continued)

Number Dec Hex		Mnemonic	Definition
13	0D	DRV_ALLRESET	Reset all disk drives.
31	1F	DRV_DPB	Return the segment and offset address of the Disk Parameter Block for the default disk of the calling process.
48	30	DRV_FLUSH	Write internal pending blocking/deblocking data buffers to disk.
39	27	DRV_FREE	Relinquish access to specified drives.
25	19	DRV_GET	Return the default drive of the calling process.
101	65	DRV_GETLABEL	Return the directory label data byte for the specified drive.
24	18	DRV_LOGINVEC	Return bit map of logged-in disk drives.
37	25	DRV_RESET	Reset the specified drives.
29	1D	DRV_ROVEC	Return bit map vector of drives set to Read-Only.
14	0E	DRV_SET	Set default drive of calling process.
100	64	DRV_SETLABEL	Create or update a directory label.
28	1C	DRV_SETRO	Set the default drive to Read-Only.
46	2E	DRV_SPACE	Return unallocated space on the specified drive.

Table 6-2. (continued)

<i>Number</i> <i>Dec Hex</i>		<i>Mnemonic</i>	<i>Definition</i>
Disk File System Calls			
30	1E	F_ATTRIB	Set file attributes.
16	10	F_CLOSE	Close file.
19	13	F_DELETE	Delete file.
51	33	F_DMASEG	Set Direct Memory Access buffer segment address.
52	34	F_DMAGET	Return segment and offset address of Direct Memory Access buffer.
26	1A	F_DMAOFF	Set the Direct Memory Access offset address.
45	2D	F_ERRMODE	Set the BDOS Error mode.
42	2A	F_LOCK	Lock record within file opened in Unlocked mode.
22	16	F_MAKE	Create file.
44	2C	F_MULTISEC	Set the BDOS Multisector Count.
15	0F	F_OPEN	Open file for record access.
152	98	F_PARSE	Parse an ASCII string and initialize an FCB.
106	6A	F_PASSWD	Set the default password.

Table 6-2. (continued)

Number Dec Hex		Mnemonic	Definition
36	24	F_RANDREC	Set the Random Record field in the FCB from the sequential record position.
20	14	F_READ	Read records sequentially.
33	21	F_READRAND	Read random records.
23	17	F_RENAME	Rename file.
17	11	F_SFIRST	Search for first matching directory FCB that matches the specified FCB.
35	23	F_SIZE	Return the size of a file.
18	12	F_SNEXT	Search for next matching directory FCB that matches the FCB specified in the F_SFIRST system call.
102	66	F_TIMEDATE	Return file's date and time stamps and password mode.
99	63	F_TRUNCATE	Truncate file to the specified Random Record Number.
43	2B	F_UNLOCK	Remove record locks.
32	20	F_USERNUM	Set or return the default user number of the calling process.
21	15	F_WRITE	Write records sequentially.

Table 6-2. (continued)

Number Dec Hex		Mnemonic	Definition
34	22	F_WRITERAND	Write random records.
40	28	F_WRITEZF	Write random records and zero-fill any previously unallocated data blocks.
103	67	F_WRITEXFCB	Create or update file's XFCB.
List Device System Calls			
158	9E	L_ATTACH	Establish ownership of the default list device by the calling process; suspend the process until the device is available.
161	A1	L_CATTACH	Conditionally establish ownership of the default list device by the calling process; return error code if the device is unavailable.
159	9F	L_DETACH	Relinquish ownership of the default list device.
164	A4	L_GET	Return the default list device number of the calling process.
160	A0	L_SET	Change the default list device for the calling process.
5	05	L_WRITE	Write a character to the default list device.

Table 6-2. (continued)

<i>Number</i> <i>Dec Hex</i>		<i>Mnemonic</i>	<i>Definition</i>
112	70	L_WRITEBLK	Write the specified number of characters (block) to the default list device.
MP/M Compatible Memory Allocation System Calls			
128	80	M_ALLOC	Allocate the memory segment between the sizes specified in the Memory Parameter Block to the calling process.
129	81	same as 128	
130	82	M_FREE	Free the specified memory segment.
CP/M Compatible Memory Allocation System Calls			
56	38	MC_ABSALLOC	Allocate a specified amount of RAM, as above, but beginning at a specific address.
54	36	MC_ABSMAX	Allocate the maximum amount of RAM available at a specified address.
58	3A	MC_ALLFREE	Free all memory owned by the calling process.
55	37	MC_ALLOC	Allocate a segment of RAM, as specified in the Memory Control Block, to the calling process.
57	39	MC_FREE	Free an area of RAM beginning at a specified address, and extending to the end of the previously-allocated memory area.



Table 6-2. (continued)

Number Dec Hex		Mnemonic	Definition
53	35	MC_MAX	Allocate the maximum amount of RAM available in the system.
Process/Program System Calls			
157	9D	P_ABORT	Terminate a process specified by name or Process Descriptor address.
47	2F	P_CHAIN	Load, initialize, and jump to the program specified in the DMA buffer.
150	96	P_CLI	Interpret and execute the specified command line by calling Command Line Interpreter (CLI).
144	90	P_CREATE	Create a subprocess.
141	8D	P_DELAY	Suspend the calling process for a specified number of system clock ticks.
142	8E	P_DISPATCH	Force a dispatch operation; give up the CPU resource to the highest priority process ready to run.
59	3B	P_LOAD	Load the specified CMD file in memory; return its base page segment address.
156	9C	P_PDADR	Return the address of the Process Descriptor of the calling process.

Table 6-2. (continued)

Number Dec Hex		Mnemonic	Definition
145	91	P_PRIORITY	Set the priority of the calling process.
151	97	P_RPL	Invoke a system call from a Resident Procedure Library.
143	8F	P_TERM	Terminate the calling process.
0	00	P_TERMCPM	Terminate the calling process.
Queue System Calls			
138	8A	Q_CREAD	Conditionally read a message from a system queue; return error code if a message is not available.
140	8C	Q_CWRITE	Conditionally write a message to a system queue; return an error code if space is not available.
136	88	Q_DELETE	Delete a system queue.
134	86	Q_MAKE	Create a system queue.
135	87	Q_OPEN	Open a system queue for subsequent queue operations.
137	89	Q_READ	Read a message from a system queue; suspend calling process until message is available.
139	8B	Q_WRITE	Write a message to a system queue; suspend calling process until space becomes available.

Table 6-2. (continued)

Number Dec Hex		Mnemonic	Definition
System System Calls			
12	0C	S_BDOSVER	Return BDOS version number, CPU and operating system type.
50	32	S_BIOS	Call specified CP/M-86 BIOS character I/O routine.
163	A3	S_OSVER	Return type and version number of Concurrent CP/M-86.
107	6B	S_SERIAL	Return the Concurrent CP/M-86 system serial number.
154	9A	S_SYSDAT	Return address of the System Data Segment (Sysdat)
Time System Calls			
105	69	T_GET	Obtain the system calendar and clock, hours and minutes only.
155	9B	T_SECONDS	Return current system date and time; hours, minutes, seconds.
104	68	T_SET	Set internal system calendar and clock to specified value.

## 6.1 System Call Summary

Table 6-3 lists the Concurrent CP/M-86 system calls in summary form, including the parameters a process must pass when calling the system call, and the values the system call returns to the process.

**Table 6-3. System Call Summary**

<i>Mnemonic</i>	<i>Dec</i>	<i>Hex</i>	<i>Input Parameters</i>	<i>Returned Values</i>
C_ASSIGN	149	95	DX = .ACB	AX = Rtn Code
C_ATTACH	146	92	none	none
C_CATTACH	162	A2	none	AX = Rtn Code
C_DELIMIT	110	6E	DX = Out Delim	AL = Out Delim
C_DETACH	147	93	none	none
C_GET	153	99	none	AL = con #
C_MODE	109	6D	DX = Con Mode	AX = Con Mode
C_SET	148	94	DL = Console	none
C_RAWIO	6	6	see def	see def
C_READ	1	1	none	AL = char
C_READSTR	10	A	DX = .Buffer	see def
C_STAT	11	B	none	AL = 00/01
C_WRITE	2	2	DL = char	none
C_WRITEBLK	111	6F	DX = .CHCB	none
C_WRITESTR	9	9	DX = .Buffer	none
DEV_POLL	131	83	DL = Device	none
DEV_SETFLAG	133	85	DL = Flag	AX = Rtn Code
DEV_WAITFLAG	132	84	DL = Flag	AX = Rtn Code
DRV_ACCESS	38	26	DX = drive Vect	none
DRV_ALLOCVEC	27	1B	none	ES:AX = Alloc Addr
DRV_ALLRESET	13	D	none	see def
DRV_DPB	31	1F	none	ES: AX = DPB Addr
DRV_FLUSH	48	30	none	see def

Table 6-3. (continued)

<i>Mnemonic</i>	<i>Dec</i>	<i>Hex</i>	<i>Input Parameters</i>	<i>Returned Values</i>
DRV_FREE	39	27	DX = drive Vect	none
DRV_GET	25	19	none	AL = Cur Disk#
DRV_GETLABEL	101	65	DX = Disk #	AL = Label Data
DRV_LOGINVEC	24	18	none	AX = Login Vect.
DRV_RESET	37	25	DX = drive Vect	AL = Err Code
DRV_ROVEC	29	1D	none	AX = R/O Vect.
DRV_SET	14	E	DL = Disk Number	see def
DRV_SETLABEL	100	64	DX = .FCB	AL = Dir Code
DRV_SETRO	28	1C	none	see def
DRV_SPACE	46	2E	DL = Disk #	see def
F_ATTRIB	30	1E	DX = .FCB	see def
F_CLOSE	16	10	DX = .FCB	AL = Dir Code
F_DELETE	19	13	DX = .FCB	AL = Dir Code
F_DMAGET	52	34	none	ES:AX = DMA Addr
F_DMAOFF	26	1A	DX = .DMA	none
F_DMASEG	51	33	DX = .DMA Seg	none
F_ERRMODE	45	2D	see def	none
F_LOCK	42	2A	DX = .FCB	AL = Err Code
F_MAKE	22	16	DX = .FCB	AL = Dir Code
F_MULTISEC	44	2C	DL = # of Sectors	AL = Rtn Code
F_OPEN	15	F	DX = .FCB	AL = Dir Code
F_PARSE	152	98	DX = .PFCB	see def
F_PASSWD	106	6A	DX = .Password	none
F_RANDREC	36	24	DX = .FCB	r0, r1, r2
F_READ	20	14	DX = .FCB	AL = Err Code
F_READRAND	33	21	DX = .FCB	AL = Err Code
F_RENAME	23	17	DX = .FCB	AL = Dir Code
F_SFIRST	17	11	DX = .FCB	AL = Dir Code
F_SIZE	35	23	DX = .FCB	r0, r1, r2
F_SNEXT	18	12	none	AL = Dir Code
F_TIMEDATE	102	66	DX = .XFCB	AL = Dir Code
F_TRUNCATE	99	63	DX = .FCB	see def

Table 6-3. (continued)

<i>Mnemonic</i>	<i>Dec</i>	<i>Hex</i>	<i>Input Parameters</i>	<i>Returned Values</i>
F_UNLOCK	43	2B	DX = .FCB	AL = Err Code
F_USERNUM	32	20	see def	see def
F_WRITE	21	15	DX = .FCB	AL = Err Code
F_WRITERAND	34	22	DX = .FCB	AL = Err Code
F_WRITEXFCB	103	67	DX = .XFCB	AL = Dir Code
F_WRITEZF	40	28	DX = .FCB	AL = Err Code
L_ATTACH	158	9E	none	none
L_CATTACH	161	A1	none	AX = Rtn Code
L_DETACH	159	9F	none	none
L_GET	164	A4	none	AL = list #
L_SET	160	A0	DL = List #	none
L_WRITE	5	5	DL = char	none
L_WRITEBLK	112	70	DX = .CHCB	none
M_ALLOC	128	80		
M_ALLOC	129	81	DX = .MPB	AX = Rtn Code
M_FREE	130	82	DX = .MPB	none
MC_ABSALLOC	56	38	DX = .MCB	see def
MC_ABSMAX	54	36	DX = .MCB	see def
MC_ALLFREE	58	3A	none	none
MC_ALLOC	55	37	DX = .MCB	see def
MC_FREE	57	39	DX = .MCB	see def
MC_MAX	53	35	DX = .MCB	see def
P_ABORT	157	9D	DX = .ABP	AX = Rtn Code
P_CHAIN	47	2F	see def	none
P_CLI	150	96	DX = .CLBUF	none
P_CREATE	144	90	DX = .PD	none
P_DELAY	141	8D	DX = #ticks	none
P_DISPATCH	142	8E	none	none

Table 6-3. (continued)

<i>Mnemonic</i>	<i>Dec</i>	<i>Hex</i>	<i>Input Parameters</i>	<i>Returned Values</i>
P_LOAD	59	3B	DX = .FCB	AX = B.P.Seg
P_PDADR	156	9C	none	ES:AX = PD Addr
P_PRIORITY	145	91	DL = Priority	none
P_RPL	151	97	DX = .CPB	AX = result
P_TERM	143	8F	DL = Term. Code	AX = Rtn Code
P_TERMCPM	0	0	none	AX = Rtn Code
Q_CREAD	138	8A	DX = .QPB	AX = Rtn Code
Q_CWRITE	140	8C	DX = .QPB	AX = Rtn Code
Q_DELETE	136	88	DX = .QPB	AX = Rtn Code
Q_MAKE	134	86	DX = .QD	none
Q_OPEN	135	87	DX = .QPB	AX = Rtn Code
Q_READ	137	89	DX = .QPB	none
Q_WRITE	139	8B	DX = .QPB	none
S_BDOSVER	12	C	none	AX = Version#
S_BIOS	50	32	DX = .BD	AX = BIOS rtn
S_OSVER	163	A3	none	AX = Version #
S_SERIAL	107	6B	DX = .serialnmb	serial nmb set
S_SYSDAT	154	9A	none	ES:AX = Sys Data Addr
T_GET	105	69	DX = .TOD	none
T_SECONDS	155	9B	DX = .TOD	none
T_SET	104	68	DX = .TOD	none

**Note:** system calls 3, 4, 7, and 8 are not supported by Concurrent CP/M-86.

Conventions used in Table 6-3:

Abs	= Absolute	Pswd	= Password
Addr	= Address	Proc	= Process
BP	= Base Page	PD	= Process Descriptor
Char	= ASCII Character	Rec	= Record
Comm	= Command	Reloc	= Relocatable
Con	= Console	Rqst	= Request
Cond.	= Conditional	Rtn	= Return
Ct	= Count	Sp	= Space
Dir	= Directory	Spec.	= Specified
Err	= Error	Sys	= System
#	= Number	Term.	= Termination
		Vect	= Vector

Upper-case mnemonics refer to Data Structures; see the function definition. A "." before a Data Structure means the byte offset of the Data Structure. A Return Code is either 0 for success or 0FFFFH to indicate failure. When the Return Code in AX is 0FFFFH, CX is the Error Code (see Table 6-5). An error code returned in AL is specific to the BDOS system call that was made.

**Table 6-4. Data Structures Index**

<i>Figure</i>	<i>Title</i>	<i>Page</i>
1-1	Concurrent CP/M-86 Virtual/Physical Environments	1-1
1-2	Concurrent CP/M-86 Functional Modules	1-3
2-1	FCB - File Control Block	2-11
2-2	Directory Label Format	2-19
2-3	XFCB - Extended File Control Block	2-21
2-4	Directory Record with SFCB	2-25
2-5	SFCB Subfields	2-26
2-6	Disk System Reset	2-45



Table 6-4. (continued)

<i>Figure</i>	<i>Title</i>	<i>Page</i>
3-1	CMD File Header Format	3-2
3-2	Group Descriptor Format	3-3
3-3	Concurrent CP/M-86 Base Page Values	3-5
4-1	Concurrent CP/M-86 8080 Memory Model	4-3
4-2	Concurrent CP/M-86 Small Memory Model	4-5
4-3	Concurrent CP/M-86 Compact Memory Model	4-6
4-4	Intel Hexadecimal File Formats	4-12
5-1	8080 and Small RSP Models	5-2
5-2	RSP Header Format	5-3
5-3	RSP Command Queue Message	5-5
5-4	RSP Data Segment	5-7
6-1	ACB - Assign Control Block	6-22
6-2	Console Buffer Format	6-34
6-3	Drive, R/O, or Login Vector Structure	6-43
6-4	DPB - Disk Parameter Block	6-50
6-5	Disk Free Space Field Format	6-66
6-6	PFCB - Parse Filename Control Block	6-91
6-7	MCB - Memory Control Block	6-135
6-8	MPB - Memory Parameter Block	6-136
6-9	MFPB - M_FREE Parameter Block	6-139
6-10	APB - Abort Parameter Block	6-146
6-11	CLI Command Line Buffer	6-150
6-12	PD - Process Descriptor	6-154
6-13	UDA - User Data Area	6-159
6-14	CPB - Call Parameter Block	6-167
6-15	QPB - Queue Parameter Block	6-171

Table 6-5. CX Error Code Reports

<i>Dec</i>	<i>Hex</i>	<i>Error Report</i>
0	00H	No error
1	01H	System call not implemented
2	02H	Illegal system call number
3	03H	Cannot find memory
4	04H	Illegal flag number
5	05H	Flag overrun
6	06H	Flag underrun
7	07H	No unused Queue Descriptors
8	08H	No free queue buffer
9	09H	Cannot find queue
10	0AH	Queue in use
12	0CH	No free process descriptors
13	0DH	No queue access
14	0EH	Empty queue
15	0FH	Full queue
18	12H	No unused Memory Descriptors
19	13H	Illegal console number
20	14H	No Process Descriptor match
21	15H	No console match
22	16H	No CLI process
23	17H	Illegal disk number
24	18H	Illegal filename
25	19H	Illegal filetype
26	1AH	Character not ready
27	1BH	Illegal memory descriptor
28	1CH	Bad return from BDOS load
29	1DH	Bad return from BDOS read
30	1EH	Bad return from BDOS open
31	1FH	Null command
32	20H	Not owner of resource
33	21H	No CSEG in load file

Table 6-5. (continued)

<i>Dec</i>	<i>Hex</i>	<i>Error Report</i>
34	22H	Process Descriptor exists on Thread Root
35	23H	Could not terminate process
36	24H	Cannot attach to process
37	25H	Illegal list device number
38	26H	Illegal password
40	28H	External termination occurred
41	29H	Fixup error upon load

## 6.2 Concurrent CP/M-86 System Calls

This section presents detailed information on the Concurrent CP/M-86 system calls. Read the entire section through before attempting to use the system calls in a program, as many of them interact with one another.

## 6.2.1 Console I/O System Calls

C_ASSIGN	
Assign Default Console Device To Another Process	
Entry Parameters:	
Register CL:	095H (149)
DX:	ACB Address - Offset
DS:	ACB Address - Segment
Returned Values:	
Register AX:	0 if assign 'OK' 0FFFFH on Failure
BX:	Same as AX
CX:	Error Code

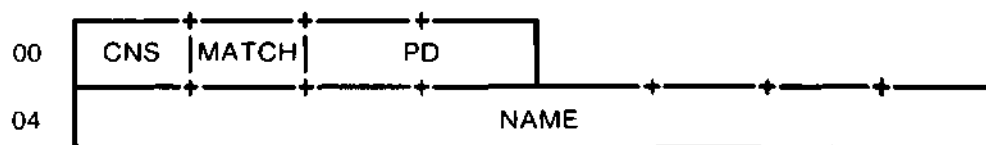


Figure 6-1. ACB - Assign Control Block

Table 6-6. ACB Field Definitions<sup>†</sup>

<i>Field</i>	<i>Definitions</i>
CNS	Console to assign
MATCH	Boolean; if 0FFH, the process being assigned the console must have the CNS as its default console for a successful Assign. IF 0H, no check is made.
PD	Process ID of the process being assigned the console. If this field is zero, a search is made of the Thread list for a process whose name is NAME. This field must be either zero or a valid Process ID. If this value is not a valid PD, an error occurs.
NAME	8-byte process name to search for. An error occurs if a process by this name does not exist.

The C\_ASSIGN system call directly assigns the specified console to a specified process. This system call overrides the normal mechanism of the C\_ATTACH and C\_DETACH system calls. The system call returns an error code if a process other than the calling process owns the console. The system call ignores other processes waiting to attach to the specified console, and they continue to wait until the current owner either calls the C\_DETACH system call, or terminates.

Refer to Table 6-5 for a list of error codes returned in CX.

C_ATTACH
Attach Default Console To Calling Process
Entry Parameters: Register CL: 092H (146)

The C\_ATTACH system call attaches the default console to the calling process. If the console is already owned by the calling process or if it is not owned by another process, the C\_ATTACH system call immediately returns with ownership established and verified. If another process owns the console, the calling process waits until the console becomes available.

Refer to Table 6-5 for a list of error codes returned in CX.

C_CATTACH	
Conditionally Attach To The Default Console	
Entry Parameters:	
Register CL:	0A2H (162)
Returned Values:	
Register AX:	0 if attach 'OK' 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The C\_CATTACH system call attaches the default console of the calling process only if the console is currently unattached.

If the console is currently attached to another process, the system call returns a value of 0FFH indicating that the console could not be attached. The system call returns a value of 0 to indicate that either the console is already attached to the process or that it was unattached and a successful attach operation was made.

Refer to Table 6-5 for a list of error codes returned in CX.

C_DELIMIT	
Set or Return Output Delimiter	
Entry Parameters:	
Register	CL: 06EH (110)
	DX: 0FFFFH (get) or
	DL: Output Delimiter (set)
Returned Values:	
Register	AL: Output Delimiter or (no value)
	BL: Same as AL

A program can set or interrogate the current Output Delimiter by calling C\_DELIMIT. If register DX = 0FFFFH, then the current Output Delimiter is returned in register AL. Otherwise, C\_DELIMIT sets the Output Delimiter to the value in register DL.

C\_DELIMIT sets the string delimiter for C\_WRITESTR. When a new process is created, the default delimiter value is set to a dollar sign, \$. The default delimiter is not inherited from the parent process.



C_DETACH	
Detach Default Console From Calling Process	
Entry Parameters:	
Register CL:	093H (147)
Returned Values:	
Register AX:	0 if detach 'OK' 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The C\_DETACH system call detaches the default console from the calling process. If the default console is not attached to the calling process, no action is taken. If other processes are waiting to attach to the console, the process with the highest priority attaches the console. If there is more than one process with the same priority waiting for the console, it is given to the queue writing processes on a first-come, first-serve basis.

Refer to Table 6-5 for a list of error codes returned in CX.

C_GET	
Return The Calling Process's Default Console	
Entry Parameters:	
Register CL:	099H (153)
Returned Values:	
Register AL:	Console number
BL:	Same as AL

The C\_GET system call returns the default console number of the calling process.

C_MODE	
Set or Return Console mode	
Entry Parameters:	
Register CL:	06DH (109)
DX:	0FFFFH (get) or Console mode (Set)
Returned Values:	
Register AX:	Console mode or (no value)
BX:	Same as AX

A process can set or interrogate the Console mode by calling C\_MODE. If register DX = 0FFFFH, then the current Console mode is returned in register AX. Otherwise, C\_MODE sets the Console mode to the value contained in register DX.

The Console mode is a 16-bit system parameter that determines the action of certain Console I/O functions. Note that the Console mode bits are numbered from right to left. The Console mode is set to zero when a new process is created; it is not inherited from its parent. The definition of the Console mode is

- bit 0 = 1 - CTRL-C only status for function 11.  
= 0 - Normal status for function 11.
- bit 1 = 1 - Disable stop scroll, CTRL-S, start scroll, CTRL-Q, support.  
= 0 - Enable stop scroll, start scroll support.
- bit 2 = 1 - Raw console output mode. Disables tab expansion for functions 2, 9, and 111. Also disables printer echo, CTRL-P, support.  
= 0 - Normal console output mode.
- bit 3 = 1 - Disable CTRL-C program termination  
= 0 - Enable CTRL-C program termination
- bit 7 = 1 - Disable CTRL-O console output byte bucket  
= 0 - Enable CTRL-O console output byte bucket

C_RAWIO	
Perform Direct Console I/O With Default Console	
<p><b>Entry Parameters:</b></p> <p>Register CL: 06H (6)</p> <p>DL: 0FFH (Input/Status) or 0FEH (Status) or 0FDH (Input) or Character (Output)</p>	
<p><b>Returned Values:</b></p> <p>Register AL: (Input/Status)</p> <p>= 0H (No Character)</p> <p>= Character</p> <p>(Status)</p> <p>= 0H - No Character</p> <p>= 0FFH - Ready</p> <p>(Input)</p> <p>= Character</p> <p>(Output)</p> <p>No return value</p> <p>BL: Same as AL</p>	

The C\_RAWIO system call allows the calling process to do Raw console I/O to its default console. Concurrent CP/M-86 verifies that the calling process owns its default console before allowing any I/O.

A process calls the C\_RAWIO system call by passing one of three different values shown in Table 6-7.

Table 6-7. C\_RAWIO Calling Values

<i>Value</i>	<i>Description</i>
0FFH	Console input status command (if no character is ready, a 00H is returned, else the character is returned.)
0FEH	Console status command (on return, register AL contains 00H if no character is ready; otherwise it contains 0FFH.)
0FDH	Console input command (if no character is ready, the calling process waits until one is typed.) Input characters are not echoed to the screen.
ASCII character	If the parameter is less than 0FDH, system call 6 assumes register DL contains a valid ASCII character and sends it to the console.

The C\_RAWIO system call places the calling process in Raw mode. The CTRL-C, CTRL-P, CTRL-S, and CTRL-O characters are not acted on by the PIN (Physical Input Process) but are passed on to the calling process when C\_RAWIO is used.

**Note:** if the virtual console is in CTRL-S mode, and the process that owns the virtual console then performs a C\_RAWIO call, the CTRL-S state is reset.

C_READ	
Read A Character From The Default Console	
Entry Parameters:	
Register CL:	01H (1)
Returned Values:	
Register AL:	Character
BL:	Same as AL

The C\_READ system call reads a character from the default console of the calling process. Before attempting the read, Concurrent CP/M-86 internally verifies the ownership of the console. If the calling process does not own the console, it relinquishes the CPU resource until the calling process can attach to the console. Typically, a process that is created through the P\_CLI system call owns its default console when it begins execution.

C\_READ echoes graphic characters read from the console. This includes the carriage return, line-feed, and backspace characters. It expands tab characters (CTRL-I) in columns of eight characters.

C\_READ ignores the termination character (CTRL-C) if the calling process cannot terminate (refer to the P\_TERM system call). C\_READ does not return until a character is typed on the console. The system suspends the calling process until a character is ready.

C_READSTR	
Read An Edited Line From The Default Console	
Entry Parameters: Register CL: 0AH (10) DX: BUFFER Address - Offset DS: BUFFER Address - Segment	

The C\_READSTR system call reads characters from the calling process's default console and places them into the specified buffer. The format of the buffer is shown in Figure 6-2. C\_READSTR performs line-editing system calls on the line as it is read from the console; it completes a line and returns upon receiving a terminator character from the console or when the maximum number of characters is reached. As in the C\_READ system call, C\_READSTR echoes all graphic characters read from the console. Concurrent CP/M-86 verifies that the calling process owns its default console before allowing I/O to begin.

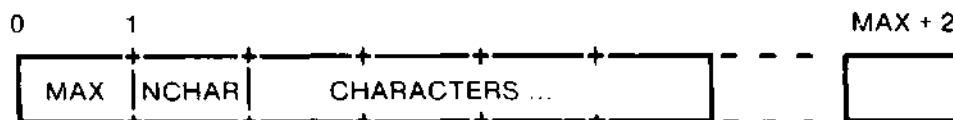


Figure 6-2. Console Buffer Format



Table 6-8. Console Buffer Field Definition

<i>Field</i>	<i>Definition</i>
MAX	Maximum number of characters that can be read into the buffer. This value must be initialized before calling the C_READSTR system call.
NCHAR	Actual number of characters read into the buffer as filled in by the C_READSTR system call.
CHARACTERS	Actual characters read from the console as filled in by the C_READSTR system call.

C\_READSTR recognizes a number of special characters used in editing the input line, as well as a set of special characters that actually control the calling process.

Table 6-9. C\_READSTR Line-editing Characters

<i>Character</i>	<i>Function</i>
RUB/DEL	Removes the last character from the line and echoes it.
(CTRL-E)	Echoes new line, a carriage return (CTRL-M), and a line-feed (CTRL-J), to the screen but does not affect the line buffer.
BACKSPACE (CTRL-H)	Removes the last character from the line and backspaces over that character.

Table 6-9. (continued)

<i>Character</i>	<i>Function</i>
TAB (CTRL-I)	Echoes enough spaces to place the next character position at a tab stop. Tab stops are fixed at every eighth character of the physical line.
LINE FEED (CTRL-J)	Terminates the input line. The C_READSTR system call does not echo a terminating character, nor does it place the character in the line buffer.
RETURN (CTRL-M)	Terminates the input line.
REDRAW (CTRL-R)	Retypes the current line after echoing a new line.
(CTRL-U)	Removes all of the current line from the line buffer, echoes a new line, and starts all over again.
(CTRL-X)	Removes all of the current line from the line buffer and echoes enough backspaces to return to the beginning of the line.

C_SET	
Set The Calling Process's Default Console	
Entry Parameters:	
Register CL:	094H (148)
DL:	Console Number
Returned Values:	
AX:	0 if successful 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The C\_SET system call changes the calling process's default console to the value specified. If the console number specified is not one supported by this particular implementation of Concurrent CP/M-86, the system call returns an error code, and does not change the default console.

Refer to Table 6-5 for a list of error codes returned in CX.

C_STAT	
Obtain The Status Of The Default Console	
Entry Parameters:	
Register CL:	0BH (11)
Returned Values:	
Register AL:	01H character ready 00H not ready
BL:	Same as AL

The C\_STAT system call checks to see if a character has been typed at the logical console (CONIN:). If the calling process is not attached to its default console, the C\_STAT system call causes a dispatch to occur and return 00H (the Not Ready condition).

This system call sets the console to the Nonraw mode, allowing recognition of special control characters such as the terminate character, CTRL-C. Use C\_RAWIO to obtain console status in Raw mode.

**Note:** if bit 0 is set in the console mode word, using the C\_MODE function call, C\_STAT only returns AL = 01H when a CTRL-C is typed on the default console.

C_WRITE	
Write A Character To The Default Console	
Entry Parameters:	
Register	CL: 02H (2)
	DL: ASCII character

The C\_WRITE system call writes the specified character to the calling process's default console. As in the C\_READ system call, Concurrent CP/M-86 verifies that the calling process owns its default console before performing the operation. On output, C\_WRITE expands tabs in columns of eight characters.

C_WRITEBLK	
Send Specified String To CONOUT:	
Entry Parameters:	
Register	CL: 06FH (111)
	DX: CHCB Address

C\_WRITEBLK sends the character string located by the Character Control Block, CHCB, addressed in register pair DX to the logical console, CONOUT:. If the Console mode is in the Default state C\_WRITEBLK expands tab characters, CTRL-I, in columns of eight characters.

The CHCB format is

bytes 0 - 1 : Offset of character string  
bytes 2 - 3 : Segment of character string  
bytes 4 - 5 : Length of character string to print

C_WRITESTR	
Print An ASCII String To The Default Console	
Entry Parameters:	
Register	CL: 09H (9)
	DX: STRING Address - Offset
	DS: STRING Address - Segment

The C\_WRITESTR system call prints an ASCII string starting at the indicated string address and continuing until it reaches a dollar sign (\$) character (024H; \$ is the default string delimiter, and can be changed by the C\_DELIMIT system call). C\_WRITESTR writes this string to the calling process's default console.

Concurrent CP/M-86 verifies that the calling process owns the console before writing the string. C\_WRITESTR sets the console to a Nonraw state and expands tabs in columns of eight characters, as does the C\_WRITE system call.

Use the C\_WRITESTR system call whenever possible, rather than the single-character system calls. The CPU overhead involved in handling the first character is the same as that for a single-character system call, but subsequent characters require as little as one-fifth the CPU overhead.

### 6.2.2 Device System Calls

DEV_POLL	
Poll A Device	
Entry Parameters:	
Register CL:	083H (131)
DL:	Device Number
Returned Values:	
Register AX:	0 on success 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The DEV\_POLL system call is used by the XIOS to poll noninterrupt-driven devices. It should be used whenever the XIOS is waiting for a noninterrupt event. The calling process relinquishes the CPU and allows Concurrent CP/M-86 to poll the device at every dispatch. The XIOS contains routines for each polling device number. These routines are called through the DEV\_POLL system call, and they return whether the device is ready or not. When the device is ready, DEV\_POLL restores the calling process to the RUN state and returns. Upon return, the calling process knows the device is ready.

Refer to Table 6-5 for a list of error codes returned in CX.



DEV_SETFLAG	
Set A System Flag	
Entry Parameters:	
Register	CL: 085H (133) DL: Flag Number
Returned Values:	
Register	AX: 0 on success 0FFFFH on failure BX: Same as AX CX: Error Code

The DEV\_SETFLAG system call is used by interrupt routines to notify the system that a logical interrupt has occurred. A process waiting for this flag is placed back into the RUN state. If there are no processes waiting, then the next process to wait for this flag returns successfully without relinquishing the CPU. The system call detects an error if the flag has already been set, and no process has done a DEV\_WAITFLAG call to reset it.

**Note:** if a process waiting for a specific flag to be set is aborted, the next DEV\_SETFLAG call is ignored and 0FFFFH is returned in AX. If the latter case is possible, the interrupt handler should continue to set call DEV\_SETFLAG until it successfully sets the flag IP, and AX = 0 on return.

Refer to Table 6-5 for a list of error codes returned in CX.

DEV_WAITFLAG	
Wait For A System Flag	
Entry Parameters:	
Register CL:	084H (132)
DL:	Flag Number
Returned Values:	
Register AX:	0 on success 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The DEV\_WAITFLAG system call is used by a process to wait for an interrupt. The process relinquishes the CPU until an interrupt routine calls the DEV\_SETFLAG system call, which places the waiting process in the RUN state. When DEV\_WAITFLAG returns to the calling process, the interrupt has occurred, or an error has occurred. An error occurs when a process is already waiting for the flag. If the Flag was set before DEV\_WAITFLAG was called, the routine returns successfully without relinquishing the CPU. This routine is meant to be used by the XIOS. The mapping between types of interrupts and flag numbers is maintained in the XIOS, although Concurrent CP/M-86 reserves flags 0, 1, 2, and 3 for system use.

Refer to Table 6-5 for a list of error codes returned in CX.

### 6.2.3 Disk Drive System Calls

The Drive Vector, Read-Only Vector, and Login Vectors are data structures referenced or returned by several Concurrent CP/M-86 Disk Drive system calls. The Drive, R/O, or Login Vectors are 16-bit values specifying one or more drives, where the least significant bit corresponds to drive A, and the high-order bit corresponds to the sixteenth drive, labeled P. The format of the Drive, R/O, and Login Vectors is illustrated below:

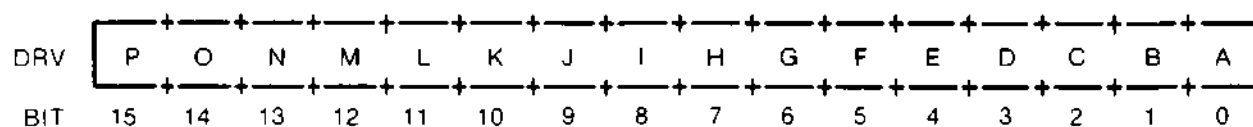


Figure 6-3. Drive, R/O, or Login Vector Structure

DRV_ACCESS	
Access Specified Disk Drives	
Entry Parameters:	
Register CL:	026H (38)
DX:	Drive Vector
Returned Values:	
AL:	Return Code
AH:	Extended Error
BX:	Same as AX

The DRV\_ACCESS system call inserts a special open file item into the system Lock List for each specified drive. While the item exists in the Lock List, the drive cannot be reset by another process. The calling process passes the drive vector in register DX. The format of the drive vector is discussed at the beginning of Section 6.2.3.

The DRV\_ACCESS system call inserts no items if insufficient free space exists in the Lock List to support all the new items or if the number of items to be inserted puts the calling process over the Lock List open file maximum. If the BDOS Error mode is the in default mode (refer to the F\_ERRMODE system call), the file system displays a message at the console identifying the error and terminates the calling process. Otherwise, DRV\_ACCESS returns to the calling process with register AL set to 0FFH and register AH set to one of the following hexadecimal values.

0AH - Open File Limit Exceeded

0BH - No Room in System Lock List

On successful calls, DRV\_ACCESS returns with register AL set to 00H.

DRV_ALLOCVEC	
Get Allocation Vector Address For The Calling Process's Default Disk	
Entry Parameters:	
Register CL:	01BH (27)
Returned Values:	
Register AX:	ALLOC Address - Offset
BX:	Same as AX
ES:	ALLOC Address - Segment

Concurrent CP/M-86 maintains an allocation vector in memory for each active disk drive. Some programs use the information provided by the allocation vector to determine the amount of free data space on a drive. Note, however, that the allocation information can be inaccurate if the drive has been marked Read-Only.

The DRV\_ALLOCVEC system call returns the address of the allocation vector for the currently selected drive. If a physical error is encountered when the BDOS Error mode is in one of the return modes (refer to the F\_ERRMODE system call), DRV\_ALLOCVEC returns the value 0FFFFH in AX.

You can use the DRV\_SPACE system call to directly return the number of free 128-byte records on a drive. The Concurrent CP/M-86 utility, SHOW, finds a drive's free space by using the DRV\_SPACE system call.

DRV_ALLRESET	
Restore All Drives To Reset State	
Entry Parameters:	
Register CL:	0DH (13)
Returned Values:	
Register AL:	0 if successful 0FFH on error
BL:	Same as AL

The DRV\_ALLRESET system call restores the file system to a reset state where all the disk drives are set to Read-Write (refer to the DRV\_SETRO and DRV\_ROVEC system calls), the default disk is set to drive A, and the default DMA address is reset to offset 080H relative to the current DMA segment address. This system call can be used, for example, by an application program that requires disk changes during operation. You can also use the DRV\_RESET system call for this purpose.

This system call is conditional under Concurrent CP/M-86. If another process has a file open on any of the drives to be reset, and the drive is also Read-Only or removable, the DRV\_ALLRESET system call is denied, and none of the specified drives are reset (see Section 2.17).

Upon return, if the reset operation is successful, DRV\_ALLRESET sets register AL to 00H. Otherwise, it sets register AL to 0FFH. If the BDOS is not in one of the return error modes (refer to the F\_ERRMODE system call), the file system displays an error message at the console identifying the process owning the first open file that caused the DRV\_ALLRESET to be denied.

DRV_DPB	
Return Address Of Disk Parameter Block For Calling Process's Default Disk	
Entry Parameters:	
Register CL:	01FH (31)
Returned Values:	
Register AX:	DPB Address - Offset 0FFFFH - on Physical Error
BX:	Same as AX
ES:	DPB Address - Segment

DRV\_DPB returns the address of the XIOS-resident Disk Parameter Block (DPB) for the currently selected drive. The calling process can use this address to extract the disk parameter values.

If a physical error is encountered when the BDOS Error mode is one of the Return Error modes (refer to the F\_ERRMODE system call), DRV\_DPB returns the value 0FFFFH.

The Disk Parameter Block (DPB) contains the parameters that define the actual disk.

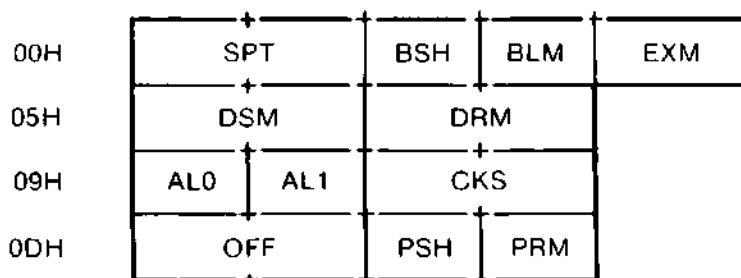


Figure 6-4. DPB - Disk Parameter Block

Table 6-10. DPB Field Definitions

Field	Definition
SPT Sectors Per Track	The number of Sectors Per Track equals the total number of physical sectors per track. Physical sector size is defined by PSH and PRM described below.
BSH Allocation Block Shift Factor	
BLM Allocation Block Mask	The data allocation block size determines the values of the data allocation Block Shift Factor and the allocation Block Mask. The Block Shift factor equals the logarithm base two of the block logical size in 128 byte records, or $BSH = \text{LOG2}(BLS)$ . The Block Mask equals the number of 128-byte records in an allocation block minus 1, or $BLM = (2^{*}BSH)-1$ . Refer to the <i>Concurrent CP/M-86 System Guide</i> for valid block sizes and BSH and BLM values.



Table 6-10. (continued)

<i>Field</i>	<i>Definition</i>
EXM Extent Mask	The data block allocation size and the number of disk allocation blocks determine the value of the Extent Mask. The Extent Mask determines the maximum number of 16k extents that can be contained in a directory entry. It is equal to the maximum number of 16K extents per directory entry minus one. Refer to the <i>Concurrent CP/M-86 System Guide</i> for EXM values.
DSM Disk Storage Maximum	The Disk Storage Maximum defines the total storage capacity of the drive. This is equal to the total number of allocation blocks minus 1 for the drive. DSM must be less than or equal to 7FFFH. If the disk uses 1024 byte blocks (BSH = 3, BLM = 7), DSM must be less than or equal to 00FFH.
DRM Directory Maximum	The Directory Maximum defines the total number of directory entries for the drive. This is equal to the total number of directory entries, minus 1, that can be kept on this drive. The directory requires 32 bytes of disk per entry. The maximum directory allocation is 16 blocks, where the block size is determined by BSH and BLM.
AL0 Directory Allocation Vector 0 AL1 Directory Allocation Vector 1	The Directory Allocation Vectors determine the reserved directory allocation blocks.

Table 6-10. (continued)

Field	Definition
CKS Checksum Vector Size	<p>The Checksum Vector Size determines the required length of the directory checksum vector and the number of directory entries that the BDOS will checksum. The Checksum Vector Size is equal to the number of directory entries divided by 4, or <math>CKS = (DRM + 1)/4</math>. If the media is fixed, CKS might be zero, no storage needs to be reserved, and the BDOS does not calculate directory checksums for the drive.</p> <p>The high-bit of CKS (that is, <math>\geq 08000H</math>) is set if the referenced drive is considered to be a nonremovable media drive. Note that this modifies the rules for resetting the drive. For more information, refer to Section 2.15.</p>
OFF Track Offset	<p>The Track Offset is the number of reserved tracks at the beginning of the disk. OFF is equal to the track number on which the directory starts.</p>
PSH Physical Record Shift Factor	<p>The Physical Record Shift Factor ranges from 0 to 5, corresponding to physical record sizes of 128, 256, 512, 1K, 2K, or 4K bytes. It is equal to the logarithm base two of the physical record size divided by 128, or <math>\text{LOG2}(\text{sector\_size}/128)</math>.</p>
PRM Physical Record Mask	<p>The Physical Record Mask ranges from 0 to 31, corresponding to physical record sizes of 128, 256, 512, 1K, 2K, or 4K bytes. It is equal to the physical sector size divided by 128 minus 1, or <math>(\text{sector\_size}/128)-1</math>.</p> <p>For more information on DPB parameters, refer to the <i>Concurrent CP/M-86 System Guide</i>, Section 5.4.</p>

DRV_FLUSH	
Flush Write-Deferred Buffers	
Entry Parameters:	
Register	CL: 030H (48)
	DL: Purge Flag
Returned Values:	
Register	AL: Error Flag
	AH: Permanent Error
	BX: Same as AX

The DRV\_FLUSH system call forces the write of any write-pending records contained in internal blocking/deblocking buffers. If register DL is set to 0FFH, DRV\_FLUSH also purges all active data buffers after performing the writes. Programs that provide write with read verify support need to purge internal buffers to ensure that verifying reads actually access the disk instead of returning data resident in internal data buffers. The Concurrent CP/M-86 PIP utility is an example of such a program.

Upon return, the system call sets register AL to 00H if the flush operation is successful. If a physical error is encountered, DRV\_FLUSH performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error and terminates the calling process. Otherwise, it returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk

DRV\_GETLABEL

Concurrent CP/M-86 Programmer's Guide

DRV_GETLABEL	
Return Directory Label Data Byte For The Specified Drive	
Entry Parameters:	
Register CL:	065H (101)
DL:	Drive
Returned Values:	
Register AL:	Directory Label Data Byte
AH:	Physical Error
BX:	Same as AX

The DRV\_GETLABEL system call returns the directory label data byte for the specified drive. The calling process passes the drive number in register DL with 0 for drive A, 1 for drive B, continuing through 15 for drive P in a full 16-drive system. The format of the directory label data byte is shown below:

- bit 7 - Require passwords for password protected files
- 6 - Perform access time and date stamping
- 5 - Perform update time and date stamping

DRV_LOGINVEC	
Return Bit Map Of Logged-in Disk Drives	
Entry Parameters:	
Register CL:	018H (24)
Returned Values:	
Register AX:	Login Vector
BX:	Same as AX

The DRV\_LOGINVEC system call returns the Login Vector in register AX. The Login Vector is a 16-bit value with the least significant bit corresponding to drive A, and the high-order bit corresponding to the 16th drive, drive P. A 0 bit indicates that the drive is not logged-in, while a 1 bit indicates the drive is logged in. Refer to the beginning of Section 6.2.3 for a complete description of the Login Vector.

DRV_RESET	
Reset Specified Disk Drives	
<b>Entry Parameters:</b>	
Register	CL: 025H (37) DX: Drive Vector
<b>Returned Values:</b>	
	AL: Return Code BL: Same as AL

The DRV\_RESET system call is used to programmatically restore specified removable media drives to the reset state (a reset drive is not logged in and is in Read-Write status). The passed parameter in register DX is a 16-bit vector of drives to be reset, where the least significant bit corresponds to drive A, and the high-order bit corresponds to the sixteenth drive, labeled P. Bit values of 1 indicate that the specified drive is to be reset. Refer to Section 2.17 for more information regarding the use of this system call.

This system call is conditional under Concurrent CP/M-86. If another process has a file open on any of the drives to be reset, the DRV\_RESET system call is denied, and none of the drives are reset.

Upon return, if the reset operation is successful, DRV\_RESET sets register AL to 00H. Otherwise, it sets register AH to 0FFH. If the BDOS Error mode is not in Return Error mode (refer to the F\_ERRMODE system call), the system displays an error message at the console, identifying the process owning the first open file that caused the DRV\_RESET request to be denied.

DRV_ROVEC	
Return Bit Map Of Read-Only Disks	
Entry Parameters:	
Register	CL: 01DH (29)
Returned Values:	
Register	AX: R/O Vector
	BX: Same as AX

DRV\_ROVEC returns a bit vector indicating which drives have the temporary Read-Only bit set. The Read-Only bit can only be set by a DRV\_SETRO call.

**Note:** when the file system detects a change in the media on a drive, it automatically logs in the drive and sets it to Read-Write.

The format of the R/O Vector is analogous to that of the Login Vector. The least significant bit corresponds to drive A; the most significant bit corresponds to drive P. For a complete description of the R/O Vector, refer to the beginning of this section.

DRV_SET	
Set Calling Process's Default Disk	
Entry Parameters:	
Register CL:	0EH (14)
DL:	Selected disk
Returned Values:	
Register AL:	Error Flag
AH:	Physical Error
BX:	Same as AX

The DRV\_SET system call designates the specified disk drive as the default disk for subsequent BDOS file operations. Set the DL register to 0 for drive A, 1 for drive B, continuing through 15 for drive P. DRV\_SET also logs in the designated drive if it is currently in the reset state. Logging in a drive activates the drive's directory for file operations.

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

Upon return, register AL equal to 00H indicates the select operation was successful. If a physical error is encountered, DRV\_SET performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call).

If the BDOS Error mode is in the default mode, the system displays a message at the console, identifying the error and terminates the calling process. Otherwise, DRV\_SET returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical error codes:

01H - Disk I/O Error : permanent error

04H - Invalid Drive : drive select error



DRV_SETLABEL	
Create Or Update A Directory Label	
Entry Parameters:	
Register	CL: 064H (100)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	AH: Physical or Extended Error
	BX: Same as AX

The DRV\_SETLABEL system call creates a directory label or updates the existing directory label for the specified drive. The calling process passes the address of an FCB containing the name, type, and extent fields to be assigned to the directory label. The name and type fields of the referenced FCB are not used to locate the directory label in the directory; they are simply copied into the updated or created directory label. Byte 12 of the FCB contains the user's specification of the directory label data byte.

The definition of the directory label data byte is

- bit 7 - Require passwords for password protected files
- 6 - Perform access time and date stamping
- 5 - Perform update time and date stamping
- 4 - Perform create time and date stamping
- 0 - Assign a new password to the directory label

(Bit 0 is the least significant bit)

If the current directory label is password protected, the correct password must be placed in the first 8 bytes of the current DMA or have been previously established as the default password (refer to the F\_PASSWD system call). If bit 0 of the directory label data byte is set to 1, it indicates that a new password for the directory label has been placed in the second eight bytes of the current DMA.

The DRV\_SETLABEL system call also requires that the referenced directory contains SFCBs in order to activate date and time stamping on the drive. If an attempt is made to activate date and time stamping when no SFCBs exist, the DRV\_SETLABEL system call returns an error code and performs no action. The Concurrent CP/M-86 INITDIR utility initializes a directory for date and time stamping by placing an SFCB in every fourth entry of the directory.

Upon return, the DRV\_SETLABEL system call returns a directory code in register AL with the value 00H if the directory label create or update was successful, or 0FFH if no space existed in the referenced directory to create a directory label. It also returns 0FFH if date and time stamping was requested and the referenced directory did not contain SFCBs. Register AH is set to 00H in all of these cases.

If a physical or extended error is encountered, the DRV\_SETLABEL system call performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the file system displays a message at the console identifying the error and terminates the calling process. Otherwise, the DRV\_SETLABEL system call returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 04H - Invalid Drive : drive select error
- 07H - Password Error

DRV_SETRO	
Set Default Disk To Read-Only	
Entry Parameters:	
Register CL:	01CH (28)
Returned Values:	
Register AL:	Return Code
Register BL:	Same as AL

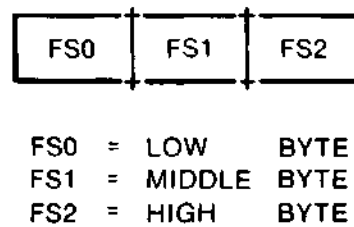
The DRV\_SETRO system call provides temporary write protection for the currently selected disk by marking the drive as Read-Only. No process can write to a disk that is in the Read-Only state. You must perform a successful DRV\_RESET operation to restore a Read-Only drive to the Read-Write state (refer to the DRV\_ALLRESET and DRV\_RESET system calls).

The DRV\_SETRO system call is conditional under Concurrent CP/M-86. If another process has an open file on the drive, the operation is denied, and the system call returns the value 0FFH to the calling process. Otherwise, it returns a 00H. If the BDOS Error mode is not in Return Error mode (refer to the F\_ERRMODE system call), the file system displays an error message at the console, identifying the process owning the first open file that caused the DRV\_SETRO request to be denied.

Note that a drive in the Read-Only state cannot be reset by a process if another process has an open file on the drive.

DRV_SPACE	
Return Free Disk Space On Specified Drive	
Entry Parameters:	
Register	CL: 02EH (46)
	DL: Drive
Returned Values:	
Register	AL: Error Flag
	AH: Physical Error
	BX: Same as AX
	First 3 bytes of DMA Buffer filled in

The DRV\_SPACE system call determines the number of free sectors (128-byte records) on the specified drive. The calling process passes the drive number in register DL, with 0 for drive A, 1 for B, continuing through 15 for drive P. DRV\_SPACE returns a binary number in the first 3 bytes of the current DMA buffer. This number is returned in the format shown in Figure 6-5.



**Figure 6-5. Disk Free Space Field Format**

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

Upon return, DRV\_SPACE sets register AL to 00H, indicating the operation was successful. However, if the BDOS Error mode is one of the return modes (refer to the F\_ERRMODE system call), and a physical error occurs, it sets register AL to 0FFH, and register AH to one of the following values:

- 01H - Disk I/O Error : permanent error
- 04H - Invalid Drive : drive select error

#### 6.2.4 File-Access System Calls

Most file-access system calls reference a File Control Block (FCB). This data structure is illustrated in Table 2-1. Refer to Section 2.4 for a comprehensive explanation of the FCB data structure, its initialization, and usage.

F_ATTRIB	
Set The Attributes Of A Disk File	
Entry Parameters:	
Register	CL: 01EH (30)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	BL: Same as AL

By calling the F\_ATTRIB system call, a process can modify a file's attributes and set its last record byte count. Other BDOS system calls can interrogate these file parameters, but only F\_ATTRIB can change them. The file attributes that can be set or reset by F\_ATTRIB are F1' through F4', Read-Only (T1'), System (T2'), and Archive (T3'). The specified FCB contains a filename with the appropriate attributes set or reset. The calling process must ensure that it does not specify an ambiguous filename. Also, if the specified file is password protected, the correct password must be placed in the first eight bytes of the current DMA buffer or have been previously established as the default password (refer to the F\_PASSWD system call).

Interface attribute F5' specifies whether an extended file lock is to be maintained after the F\_ATTRIB call. Interface attribute F6' specifies if the specified file's byte count is to be set. The interface attribute definitions are listed below:

- F5' = 0 - Do not maintain an extended file lock (default)
- F5' = 1 - Maintain an extended file lock
- F6' = 0 - Do not set byte count (default)
- F6' = 1 - Set byte count

If F5' is set and the referenced FCB specifies a file with an extended file lock, the calling process maintains the lock on the file. Otherwise, the file becomes available to other processes on the system. Section 2.11 describes extended file locking in detail.

If interface attribute F6' is set, the calling process must set the CR field of the referenced FCB to the new byte count value. A process can access a file's byte count value with the BDOS F\_OPEN, F\_SFIRST, and F\_SNEXT system calls. File byte counts are described in Section 2.15.

F\_ATTRIB searches the FCB specified directory for an entry belonging to the current user number that matches the FCB specified name and type fields. The system call then updates the directory to contain the selected indicators, and if interface attribute F6' is set, the specified byte count value. Note that the last record byte count is maintained in the byte 13 of a file's directory FCBs.

File attributes T1', T2', and T3' are defined by Concurrent CP/M-86 as described in Section 2.4.2. Attributes F1' thru F4' of command files are defined as compatibility attributes, as described in Section 2.12. However, for all other files, attributes F1' through F4' are available for definition by the user. Attributes F5' through F8' are reserved as interface attributes and cannot be used as file attributes. Interface attributes are described in Section 2.4.3.

An F\_ATTRIB system call is not performed if the referenced FCB specifies a file currently open for another process. It is performed, however, if the referenced file is open by the calling process in Locked mode. However, the file's lock entry is purged when this is done and the file system prevents continued read and write operations on the file. F\_ATTRIB does not set the attributes of a file currently open in Read-Only or Unlocked mode for any process.



Making an F\_ATTRIB system call for an open file can adversely affect the performance of the calling process. For this reason, you should close an open file before you call the F\_ATTRIB system call.

Upon return, F\_ATTRIB returns a directory code in register AL with the value 00H if the system call is successful, or 0FFH if the file specified by the referenced FCB is not found. Register AH is set to 00H in both cases.

If a physical or extended error is encountered, the F\_ATTRIB system call performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the file system displays a message at the console identifying the error and terminates the process. Otherwise, it returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 04H - Invalid Drive : drive select error
- 05H - File open by another process
- 07H - Password Error
- 09H - Illegal ? in FCB

F_CLOSE	
Close A Disk File	
Entry Parameters:	
Register	CL: 010H (16)
	DX: FCB Address - Offset
-	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	AH: Physical or Extended Error
	BX: Same as AX

The F\_CLOSE system call performs the inverse operation of the F\_OPEN system call. The referenced FCB must have been previously activated by a successful F\_OPEN or F\_MAKE system call. Interface attributes F5' and F6' specify how the file is to be closed, as shown below:

- F5' = 0, F6' = 0 - Default Close
- F5' = 0, F6' = 1 - Extend File Lock
- F5' = 1, F6' = 0 - Partial Close
- F5' = 1, F6' = 1 - Partial Close

The F\_CLOSE system call performs the following steps regardless of the interface attribute specification. First of all, it verifies that the referenced FCB has a valid checksum. If the checksum is invalid, F\_CLOSE performs no action and returns an error code.

If the checksum is valid and the referenced FCB contains new information because of write operations to the FCB, F\_CLOSE permanently records the new information in the directory. If the FCB does not contain new information, the directory update step is bypassed. However, F\_CLOSE always attempts to locate the FCB's corresponding entry in the directory and returns an error code if the directory entry cannot be found.

If the F\_CLOSE system call successfully performs the above steps, it performs different actions, depending on how the interface attributes are set. In default close operations, F\_CLOSE decrements the file's open count, which is maintained in the file's system Lock List entry. If the open count decrements to zero, it indicates that the number of default close operations for the file matches the number of open operations.

If the open count decrements to zero, F\_CLOSE permanently closes the file by performing the following steps. First of all, it removes the file's item from the system Lock List. If the FCB is opened in Unlocked mode, it also purges all record locks belonging to the file from the system Lock List. In addition, F\_CLOSE invalidates the FCB's checksum to ensure the referenced FCB is not subsequently used with BDOS system calls that require an open FCB (for example, F\_WRITE).

If the open count does not decrement to zero, F\_CLOSE simply returns to the calling process and the file remains open.

For partial close operations, F\_CLOSE does not decrement the file's open count and returns to the calling process. The file always remains open following a partial close request.

Closing a file with an extended file lock modifies the way F\_CLOSE performs a permanent close. F\_CLOSE only honors an extended lock request on a permanent close of a file opened in Locked mode. If these conditions are satisfied, F\_CLOSE invalidates the FCB's checksum but maintains the lock item. Thus, although the file is permanently closed, other processes cannot access the file. Section 2.11 describes extended file locking in detail.

Upon return, the F\_CLOSE system call returns a directory code in register AL with the value 00H if the close operation is successful, or 0FFH if the file is not found. Register AH is set to 0 in both of these cases.

If a physical or extended error is encountered, the F\_CLOSE system call performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the file system displays a message identifying the error at the console and terminates the calling process. Otherwise the F\_CLOSE system call returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 04H - Invalid Drive : drive select error
- 06H - Close Checksum Error

F_DELETE	
Delete A Disk File	
Entry Parameters:	
Register	CL: 013H (19)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	AH: Physical or Extended Error
	BX: Same as AX

The F\_DELETE system call removes files and/or XFCBs that match the FCB addressed in register DX. The filename and filetype fields can contain wildcard file specifications (question marks in bytes 1 through 11), but byte 0 cannot be a wildcard as it can be in the F\_SFIRST and F\_SNEXT system calls. Interface attribute F5' specifies the type of delete operation to be performed, as shown below:

F5' = 0 - Standard Delete (Default mode)

F5' = 1 - Delete only XFCB's and maintain an extended file lock.

If any of the files specified by the referenced FCB are password protected, the correct password must be placed in the first eight bytes of the current DMA buffer or it must have been previously established as the default password (refer to the F\_PASSWD system call).

For standard delete operations, the F\_DELETE system call removes all directory entries belonging to files that match the referenced FCB. All disk directory and data space owned by the deleted files is returned to free space and becomes available for allocation to other files. Directory XFCBs that were owned by the deleted files are also removed from the directory. If interface attribute F5' of the FCB is set to 1, F\_DELETE deletes only the directory XFCBs matching the referenced FCB.

**Note:** if any of the files matching the input FCB specification fail the password check, are Read-Only, or are currently open by another process, then F\_DELETE deletes no files or XFCBs. This applies to both types of delete operations.

Interface attribute F5' also specifies whether an extended file lock is to be maintained after the F\_DELETE call. If F5' is set and the referenced FCB specifies a file with an extended lock, the calling process maintains the lock on the file. Section 2.11 describes extended file locking in detail.

A process can delete a file that it currently has open if the file is opened in Locked mode. However, the BDOS returns a checksum error if the process makes a subsequent reference to the file with a BDOS system call requiring an open FCB. A process cannot delete files open in Read-Only or Unlocked mode.

Deleting an open file can adversely affect the performance of the calling process. For this reason, you should close an open file before you delete it.

Upon return, the F\_DELETE system call returns a directory code in register AL with the value 00H if the delete is successful, or 0FFH if no file matching the referenced FCB is found. Register AH is set to 0 in both of these cases. If a physical or extended error is encountered, F\_DELETE performs different actions, depending on the BDOS Error mode (refer to the F\_ERRMODE system call).

If the BDOS Error mode is the default mode, the system displays a message identifying the error at the console and terminates the calling process. Otherwise, it returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 03H - Read/Only File
- 04H - Invalid Drive : drive select error
- 05H - File opened by another process or open  
in Read-Only or Unlocked mode
- 07H - Password Error

F_DMAGET	
Return Address Of Direct Memory Access Buffer	
<b>Entry Parameters:</b> Register CL: 034H (52)	
<b>Returned Values:</b> Register AX: DMA Offset BX: Same as AX ES: DMA Segment	

F\_DMAGET returns the current DMA Base Segment address in ES, with the current DMA Offset in DX.



F_DMAOFF	
Set The Direct Memory Access Offset	
Entry Parameters:	
Register	CL: 01AH (26)
	DX: DMA Address - Offset

DMA is an acronym for Direct Memory Access, which is often used with disk controllers that directly access the memory of the computer to transfer data to and from the disk subsystem. Under Concurrent CP/M-86, the current DMA is usually defined as the buffer in memory where a record resides before a disk write and after a disk read operation. If the BDOS Multisector Count is equal to one (refer to the F\_MULTISEC system call), the size of the buffer is 128 bytes. However, if the BDOS Multisector Count is greater than one, the size of the buffer must equal  $N * 128$ , where N equals the Multisector Count.

Some BDOS system calls also use the current DMA to pass parameters and to return values. For example, BDOS system calls that check and assign file passwords require that the password be placed in the current DMA buffer. As another example, DRV\_SPACE returns its results in the first 3 bytes of the current DMA. When the current DMA is used in this context, the size of the buffer in memory is determined by the specific requirements of the system call.

When the P\_CLI system call initiates a transient program, it sets the DMA offset to 080H and the DMA Segment or Base to its initial Data Segment. DRV\_ALLRESET also sets the DMA offset to 080H. The F\_DMAOFF system call can change this default value to another memory address. The DMA address remains at its current value until it is changed by an F\_DMASEG, F\_DMAOFF, or DRV\_ALLRESET call.

F_DMASEG	
Set Direct Memory Access Segment Address	
Entry Parameters:	
Register	CL: 033H (51)
	DX: DMA Segment Address

F\_DMASEG sets the segment value of the current DMA buffer address. The word parameter in DX is a paragraph address and is used with the DMA offset value to specify the 20-bit address of the DMA buffer. Refer to the F\_DMAOFF system call for additional information.

Note that upon initial program loading, the default DMA base is set to the address of the user's data segment (the initial value of DS) and the DMA offset is set to 0080H, which provides access to the default buffer in the Base Page.

F_ERRMODE	
Set BDOS Error Mode For Error Returns	
Entry Parameters:	
Register	CL: 02DH (45)
	DL: BDOS Error mode

The BDOS Error mode is a system parameter maintained for each running process that determines how the file system handles physical and extended errors. Physical and extended errors are described in Section 2.18. The BDOS Error mode has three states: the default mode, Return and Error mode, and Return and Display mode.

If a physical or extended error occurs when the BDOS Error mode is in the default mode, the BDOS displays a system message at the console identifying the error and terminates the calling process.

If a physical or extended error occurs when the BDOS Error mode is in Return Error mode, the BDOS sets register AL to 0FFH, places an error code identifying the physical or extended error in register AH, and returns to the calling process.

If a physical or extended error occurs when the BDOS Error mode is in Return and Display mode, the BDOS displays the system message before returning to the calling process, and sets registers AH and AL as in the Return Error mode.

The F\_ERRMODE system call sets the BDOS Error mode for the calling process to the mode specified in register DL. If register DL is set to 0FFH, the mode is set to Return Error mode. If register DL is set to 0FEH, the mode is set to Return and Display mode. If register DL is set to any other value, the mode is set to the default mode.

F_LOCK	
Lock Records In A Disk File	
Entry Parameters:	
Register	CL: 02AH (42)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Error Code
	AH: Physical Error
	BX: Same as AX

The F\_LOCK system call allows a process to establish temporary ownership to particular records within a file. This system call is only supported for files open in Unlocked mode. If it is called for a file open in Locked or Read-Only mode, no locking action is performed and a successful result is returned. This provides compatibility between Concurrent CP/M-86 and CP/M-86.

The calling process passes the address of an FCB in which the random record field is filled with the Random Record Number of the first record to be locked. The number of records to be locked is determined by the BDOS Multisector Count (refer to the F\_MULTISEC system call.) The current DMA must also contain the 2-byte File ID returned by F\_OPEN or F\_MAKE when the referenced FCB was opened. Note that the File ID is only returned by the F\_OPEN and F\_MAKE system call when the Open mode is Unlocked.

Interface attribute F5' specifies the type of lock to perform. Interface attribute F6' specifies whether records have to exist in order to be locked. The F\_LOCK interface attribute definitions are listed below:

F5' = 0 - Exclusive lock (default)

F5' = 1 - Shared lock

F6' = 0 - Lock existing records only (default)

F6' = 1 - Lock logical records

These options are described in detail in Section 2.14.

F\_LOCK verifies that a locking conflict with another process does not exist for each of the records to be locked. In addition, if F\_LOCK is called with attribute F6' reset, it also verifies that each record number to be locked exists within the specified file. Both tests are made before any records are locked.

Most F\_LOCK requests require a new entry in the BDOS system Lock List. If there is insufficient space in the system Lock List to satisfy the lock request, or if the process record lock limit is exceeded, then F\_LOCK does not lock any records and returns an error code to the calling process.

Upon return, the F\_LOCK system call sets register AL to 00H if the lock operation is successful. Otherwise, register AL contains one of the following error codes:

- 01H - Reading unwritten data
- 03H - Cannot close current extent
- 04H - Seek to unwritten extent
- 06H - Random Record Number out of range
- 08H - Record locked by another process
- 0AH - FCB Checksum Error
- 0BH - Unlocked file verification error
- 0CH - Process record lock limit exceeded
- 0DH - Invalid File ID
- 0EH - No Room in System Lock List
- 0FFH - Physical error; refer to register AH

The system call returns error code 01H when it accesses a data block that has not been previously written.

The system call returns error code 03H when it cannot close the current extent prior to moving to a new extent.

The system call returns error code 04H when it accesses an extent that has not been created.

The system call returns error code 06H when byte 35 (R2) of the referenced FCB is greater than 3.

The system call returns error code 08H if the specified record is locked by another process with an incompatible lock type.

The system call returns error code 0AH if the referenced FCB failed the FCB checksum test.

The system call returns error code 0BH if the BDOS cannot locate the referenced FCB's directory entry when attempting to verify that the FCB contains current information.

The system call returns error code 0CH if performing the lock request would require that the process consume more than the maximum allowed number of system Lock List entries.

The system call returns error code 0DH when an invalid File ID is placed at the beginning of the current DMA.

The system call returns error code 0EH when the system Lock List is full and performing the lock request would require at least one new entry.

The system call returns error code 0FFH if a physical error is encountered, and the BDOS Error mode is either Return Error mode or Return and Display Error mode (refer to the F\_ERRMODE system call). If the Error mode is in the default mode, the system displays a message at the console identifying the physical error and terminates the calling process. When the system call returns a physical error to the calling process, it is identified by register AH as shown below:

01H - Disk I/O Error : permanent error

04H - Invalid Drive : drive select error

F_MAKE	
Create A Disk File	
Entry Parameters:	
Register	CL: 016H (22)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	AH: Physical or Extended Error
	BX: Same as AX

The F\_MAKE system call creates a new directory entry for a file under the current user number. It also creates an XFCB for the file if the referenced drive has a directory label that enables password protection on the drive, and the calling process assigns a password to the file.

The calling process passes the address of the FCB with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and byte 12 set to the extent number. Byte 12, the EX field, is usually set to 00H. Byte 32 of the FCB, the CR field, must be initialized to 00H, before or after the F\_MAKE call, if the intent is to write sequentially from the beginning of the file.



Interface attribute F5' specifies the mode in which the file is to be opened. Interface attribute F6' specifies whether a password is to be assigned to the created file. The interface attributes are summarized below:

- F5' = 0 - Open in Locked mode (default)
- F5' = 1 - Open in Unlocked mode
- F6' = 0 - Do not assign password (default)
- F6' = 1 - Assign password to created file

When attribute F6' is set to 1, the calling process 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. Note that F\_MAKE only interrogates attribute F6' if the referenced drive's directory label has enabled password support. The XFCB Password mode is summarized below:

#### XFCB Password Mode

- Bit 7 - Read mode
- Bit 6 - Write mode
- Bit 5 - Delete mode

The F\_MAKE system call returns with an error code if the referenced FCB names a file that currently exists in the directory under the current user number. If there is any possibility of duplication, an F\_DELETE call should precede the F\_MAKE call.

If the make file operation is successful, it activates the referenced FCB for record operations (opens the FCB) and initializes both the directory entry and the referenced FCB to an empty file. It also computes a checksum and assigns it to the FCB. BDOS system calls that require an open FCB (for example, F\_WRITE) verify that the FCB checksum is valid before performing their operation. If the file is opened in Unlocked mode, F\_MAKE also sets bytes R0 and R1 in the FCB to a two-byte value called the File ID. The File ID is a required parameter for the BDOS Lock Record and Unlock Record system calls. Note that the F\_MAKE system call initializes all file attributes to 0.

The BDOS file system also creates an open file item in the system Lock List to record a successful F\_MAKE operation. While this item exists, no other process can delete, rename, truncate, or set the file attributes of this file.

A creation and/or update stamp is made for the created file if the referenced drive contains a directory label that enables creation and/or update time and date stamping and the FCB extent number is equal to 0.

F\_MAKE also creates an XFCB for the created file if the referenced drive contains a directory label that enables password protection, interface attribute F6' of the FCB is 1, and the FCB is an extent zero FCB. In addition, F\_MAKE also assigns the password and password mode placed in the first nine bytes of the DMA to the XFCB.

Upon return, the F\_MAKE system call returns a directory code in register AL with the value 00H if the make operation is successful, or 0FFH if no directory space is available. Register AH is set to 00H in both cases.

If a physical or extended error is encountered, the F\_MAKE system call performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error and terminates the calling process. Otherwise, it returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 04H - Invalid Drive : drive select error
- 08H - File Already Exists
- 09H - Illegal ? in FCB
- 0AH - Open File Limit Exceeded
- 0BH - No Room in System Lock List

F_MULTISEC	
Set BDOS Multisector Count	
Entry Parameters:	
Register	CL: 02CH (44)
	DL: Number of Sectors
Returned Values:	
Register	AL: Return Code
	BL: Same as AL

The F\_MULTISEC system call provides logical record blocking under Concurrent CP/M-86. It enables a process to read and write from 1 to 128 physical records of 128 bytes at a time during subsequent BDOS read and write system calls. It also specifies the number of 128-byte records to be locked or unlocked by the F\_LOCK and F\_UNLOCK system calls.

F\_MULTISEC sets the Multisector Count value for the calling process to the value passed in register DL. Once set, the specified Multisector Count remains in effect until the calling process makes another F\_MULTISEC system call and changes the value. Note that the P\_CLI system call sets the Multisector Count to one when it initiates a transient program.

The Multisector Count affects BDOS error reporting for the BDOS read and write system calls. With the exception of physical errors, if an error occurs during these system calls when the Multisector is greater than one, they return the number of records successfully processed in register AH.

Upon return, the system call sets register AL to 00H if the specified value is in the range of 1 to 128. Otherwise, it sets register AL to 0FFH.

F_OPEN	
Open A Disk File	
Entry Parameters:	
Register	CL: 0FH (15)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	AH: Physical or Extended Error
	BX: Same as AX

The F\_OPEN system call activates the FCB for a file that exists in the disk directory under the currently active user number or user zero. The calling process passes the address of the FCB, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and byte 12 specifying the extent. Byte 12 is usually set to zero.

Interface attributes F5' and F6' of the FCB specify the mode in which the file is to be opened, as shown below:

F5' = 0, F6' = 0 - Open in Locked mode (default mode)  
F5' = 1, F6' = 0 - Open in Unlocked mode  
F5' = 0 or 1, F6' = 1 - Open in Read-Only mode

If the file is password protected in Read mode, the correct password must be placed in the first eight bytes of the current DMA or have been previously established as the default password (refer to the F\_PASSWD system call). If the current record field of the FCB, CR, is set to 0FFH, the F\_OPEN system call returns the byte count of the last record of the file in the CR field. The last record byte count for a file can be set using the F\_SETATTRIB system call.

**Note:** the calling process must set the CR field of the FCB to 00H if the file is to be accessed sequentially from the first record.

The F\_OPEN system call performs the following steps for files opened in locked or Read-Only mode. If the current user is nonzero and the file to be opened does not exist under the current user number, the Open File system call searches user 0 for the file. If the file exists under user 0 and has the system attribute (T2') set, the file is opened under user 0. The Open mode is automatically set to Read-Only when this is done.

The F\_OPEN system call also performs the following action for files opened in locked mode when the current user number is 0. If the file exists in the directory under user 0, and has both the system attribute (T2') set and the Read-Only attribute (T1') set, the Open mode is automatically set to Read-Only. Note that Read-Only mode implies the file can be concurrently accessed by other processes if they also open the file in Read-Only mode.

If the open operation is successful, F\_OPEN activates the user's FCB for record operations as follows: F\_OPEN copies the relevant directory information from the matching directory FCB into bytes D0 through D15 of the FCB. It also computes a checksum and assigns it to the FCB. All BDOS system calls that require an open FCB (for example, F\_READ) verify that the FCB checksum is valid before performing their operation.

If the file is opened in Unlocked mode, the F\_OPEN system call sets bytes R0 and R1 of the FCB to a two-byte value called the File ID. The File ID is a required parameter for the F\_LOCK and F\_UNLOCK system calls. If the Open mode is forced to Read-Only, F\_OPEN sets interface attribute F8' to 1 in the user's FCB. In addition, the system call sets attribute F7' to 1 if the referenced file is password protected in Write mode and the correct password was not passed in the DMA or did not match the default password. The BDOS does not support write operations for an activated FCB if interface attribute F7' or F8' is set to 1.

The BDOS file system also creates an open file item in the system Lock List to record a successful open file operation. While this item exists, no other process can delete, rename, or modify the file's attributes. In addition, this item prevents other processes from opening the file if the file is opened in Locked mode. It also requires that other processes match the file's Open mode if the file is opened in Unlocked or Read-Only mode. This item remains in the system Lock List until the file is permanently closed or until the process that opened the file terminates.

When the open operation is successful, the F\_OPEN system call also makes an access time and date stamp for the opened file when the following conditions are satisfied: the referenced drive has a directory label that requests access date and time stamping, the FCB extent field is equal to zero, and the referenced drive is Read-Write.

Upon return, F\_OPEN returns a directory code in register AL with the value 00H if the open is successful, or 0FFH if the file is not found. Register AH is set to 0 in both of these cases. If a physical or extended error is encountered, the F\_OPEN system call performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message identifying the error at the console and terminates the process. Otherwise, F\_OPEN returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 04H - Invalid Drive : drive select error
- 05H - File is open by another process or by the  
current process in an incompatible mode
- 07H - Password Error
- 09H - Illegal ? in FCB
- 0AH - Open File Limit Exceeded
- 0BH - No Room in System Lock List

F_PARSE	
Parse An ASCII String And Initialize An FCB	
<p>Entry Parameters:</p> <p>Register CL: 098H (152)</p> <p>DX: PFCB Address - Offset</p> <p>DS: PFCB Address - Segment</p> <p>Returned Values:</p> <p>Register AX: 0FFFFH if error</p> <p>0 if end of filename string</p> <p>end of line</p> <p>address of next item to parse</p> <p>BX: Same as AX</p> <p>CX: Error Code</p>	



Figure 6-6. PFCB - Parse Filename Control Block

Table 6-11. PFCB Field Definitions

<i>Field</i>	<i>Description</i>
FILENAME	Offset of an ASCII file specification to parse. The offset is relative to the same Data Segment as the PFCB.
FCBADR	Offset of a File Control Block to initialize. The offset is relative to the same Data Segment as the PFCB.

The F\_PARSE system call parses an ASCII file specification (FILENAME) and prepares a File Control Block (FCB). The calling process passes the address of a data structure called the Parse Filename Control Block, (PFCB) in registers DX and DS. The PFCB contains the offset of the ASCII filename string followed by the offset of the target FCB.

F\_PARSE assumes the file specification to be in the following form

{D:} FILENAME {.TYP} {;PASSWORD}

where those items enclosed in curly brackets are optional.

The F\_PARSE system call parses the first file specification it finds in the input string. First of all, it eliminates leading blanks and tabs. F\_PARSE then assumes the file specification ends on the first delimiter it encounters that is out of context with the specific field it is parsing. For instance, if it finds a colon (:), and it is not the second character of the file specification, the colon delimits the whole file specification.



The F\_PARSE 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)
	(vertical bar)
[	(left square bracket)
]	(right square bracket)

If the F\_PARSE system call encounters a nongraphic character in the range 1 through 31 not listed above, it treats the character as an error.

The F\_PARSE system call initializes the specified FCB as shown in Table 6-12.

Table 6-12. FCB Initialization

<i>Byte number</i>	<i>Explanation</i>
byte 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, etc.
byte 1-8	The name is set to the specified filename. All letters are converted to upper-case. If the name is not eight characters long, the remaining bytes in the filename field are padded with blanks. If the filename has an asterisk (*), all remaining bytes in the filename field are filled in with question marks (?). The system call returns an error if the filename is more than eight bytes long.
byte 9-11	The type is set to the specified filetype. If no type is specified, the type field is initialized to blanks. All letters are converted to upper-case. If the type is not three characters long, the remaining bytes in the filetype field are padded with blanks. If an asterisk is encountered, all remaining bytes are filled in with question marks. The system call returns an error if the type field is more than 3 bytes long.
byte 12-15	Filled in with zeros.
byte 16-23	The password field is set to the specified password. If no password is specified, this field is initialized to blanks. If the password is not eight characters long, remaining bytes are padded with blanks. All letters are converted to upper-case. The system call returns an error if the password field is more than eight bytes long.
byte 24-31	Reserved for system use.

If an error occurs, F\_PARSE returns 0FFFFH in register AX indicating the error.

On a successful parse, the F\_PARSE system call checks the next item in the FILENAME string. It scans for the first character that follows trailing blanks and tabs. If the character is a line feed (0AH), a carriage return (0DH), or a null character (00H), it returns a 0 indicating the end of the FILENAME 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 F\_PARSE system call is to be used to parse a subsequent filename in the FILENAME string, the returned address should be advanced over the delimiter before placing it in the PFCB.

Refer to Table 6-5 for a list of error codes returned in CX.

F_PASSWD	
Establish A Default Password For File Access	
Entry Parameters:	
Register	CL: 06AH (106)
	DX: Password Address - Offset
	DS: Password Address - Segment

The F\_PASSWD system call allows a process to specify a password value before a file protected by the password is accessed. When the file system accesses a password-protected file, it checks the current DMA, and the default password for the correct value. If either value matches the file's password, full access to the file is allowed.

Concurrent CP/M-86 maintains a default password for each process running on the system. A new process inherits its initial default password from its parent, the process creating the new process.

**Note:** changing the default password does not affect other processes currently running on the system.

To make an F\_PASSWD call, the calling process passes the address of an eight-byte field containing the password.

F_RANDOMREC	
Return The Random Record Number Of The Next Record To Access In A Disk File	
Entry Parameters:	
Register	CL: 024H (36)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Random Record Field of FCB Set	

The F\_RANDOMREC system call returns the Random Record Number of the next record to be accessed from a file that has been read or written sequentially to a particular point. The system call returns this value in the Random Record field, bytes R0, R1, and R2, of the addressed FCB. The F\_RANDOMREC system call can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various key fields. As each key is encountered, F\_RANDOMREC is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record number minus one is placed into a table with the key for later retrieval.

After scanning the entire file and tabularizing the keys and their record numbers, you can move directly to a particular record by performing a random read using the corresponding Random Record Number that was saved earlier. The scheme is easily generalized when variable record lengths are involved, because the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

F\_RANDOMREC can also be used when switching from a sequential read or write to a random read or write. A file is sequentially accessed to a particular point in the file, F\_RANDOMREC is called to set the record number, and subsequent random read and write operations continue from the next record in the file.

F_READ	
Read Records Sequentially From A Disk File	
<b>Entry Parameters:</b>	
Register	CL: 014H (20)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
<b>Returned Values:</b>	
Register	AL: Error Code
	AH: Physical Error
	BX: Same as AX

The F\_READ system call reads the next 1 to 128 128-byte records from a file into memory, beginning at the current DMA address. The BDOS Multisector Count (refer to the F\_MULTISEC system call) determines the number of records to be read. The default is one record. The addressed FCB must have been previously activated by an F\_OPEN or F\_MAKE system call.

F\_READ reads each record from the current record (CR) field in the FCB, relative to the current extent, then automatically increments the CR field to the next record position. If the CR field overflows, then F\_READ automatically opens the next logical extent and resets the CR field to zero for the next read operation. The calling process must set the CR field to 00H following the open call if the intent is to read sequentially from the beginning of the file.

Upon return, the F\_READ system call sets register AL to zero if the read operation is successful. Otherwise, register AL contains an error code identifying the error as shown below:

- 01H - Reading unwritten data (end-of-file)
- 08H - Record locked by another process
- 09H - Invalid FCB
- 0AH - FCB Checksum Error
- 0BH - Unlocked file verification error
- 0FFH - Physical error; refer to register AH

The system call returns error code 01H if no data exists at the next record position of the file. The no data situation is usually encountered at the end of a file. However, it can also occur if you try to read a data block that has not been previously written or an extent that has not been created. These situations are usually restricted to files created or appended with the BDOS random write system calls (F\_WRITERAND and F\_WRITEZF).

The system call returns error code 08H if the calling process attempts to read a record locked by another process with an exclusive lock. This error code is only returned for files opened in Unlocked mode.

The system call returns error code 09H if the FCB is invalidated by a previous F\_CLOSE system call that returned an error.

The system call returns error code 0AH if the referenced FCB failed the FCB checksum test.

The system call returns error code 0BH if the BDOS cannot locate the FCB's directory entry when attempting to verify that the referenced FCB contains current information. The system call only returns this error for files opened in Unlocked mode.

The system call returns error code 0FFH if a physical error is encountered and the BDOS Error mode is in one of the return modes (refer to the F\_ERRMODE system call). If the Error mode is in the default mode, the file system displays a message at the console identifying the physical error and terminates the calling process. When the system call returns a physical error to the calling process, it is identified by register AH as shown below:

01H - Disk I/O Error : permanent error

04H - Invalid Drive : drive select error

On all error returns, except for physical error returns (AL = 255), F\_READ sets register AH to the number of records successfully read before the error was encountered. This value can range from 0 to 127 depending on the current BDOS Multisector Count. It is always set to zero when the Multisector Count is equal to one.



F_READRAND	
Read Random Records From A Disk File	
Entry Parameters:	
Register	CL: 021H (33)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Error Code
	AH: Physical Error
	BX: Same as AX

The F\_READRAND system call is similar to the F\_READ system call except that the read operation takes place at a particular Random Record Number, selected 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 high byte last, R2. The Random Record Number can range from 0 to 262,143. This corresponds to a maximum value of 3 in byte R2.

To read a file with the F\_READRAND system call, the calling process must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. The base extent might or might not contain any allocated data.

The **F\_READRAND** system call reads the record specified by the random record field into the current DMA address. **F\_READRAND** automatically sets the FCB extent and current sector number values, **EX** and **CR**, but unlike the **F\_READ** system call, it does not advance the current record number. Thus, a subsequent **F\_READRAND** call rereads the same record. After a random read operation, a file can be accessed sequentially, starting from the current randomly accessed position. However, the last randomly accessed record is reread or rewritten when switching from random to sequential mode.

If the BDOS Multisector count is greater than one (refer to the **F\_MULTISEC** system call), **F\_READRAND** reads multiple consecutive records into memory beginning at the current DMA. **F\_READRAND** automatically increments the **R0**, **R1**, **R2** field of the FCB to read each record. However, it restores the FCB's Random Record Number to the first record's value upon return to the calling process.

Upon return, **F\_READRAND** sets register **AL** to **00H** if the read operation is successful. Otherwise, register **AL** contains one of the following error codes:

- 01H - Reading unwritten data
- 03H - Cannot close current extent
- 04H - Seek to unwritten extent
- 06H - Random record number out of range
- 08H - Record locked by another process
- 0AH - FCB Checksum Error
- 0BH - Unlocked file verification error
- 0FFH - Physical error refer to register **AH**

The system call returns error code **01H** when it accesses a data block not previously written. This may indicate an end-of-file (EOF) condition.

The system call returns error code **03H** when it cannot close the current extent prior to moving to a new extent.

The system call returns error code 04H when a read random operation accesses an extent that has not been created.

The system call returns error code 06H when byte 35 (R2) of the referenced FCB is greater than 3.

The system call returns error code 08H if the calling process attempts to read a record locked by another process with an exclusive lock. This error code is only returned for files opened in Unlocked mode.

The system call returns error code 0AH if the referenced FCB failed the FCB checksum test.

The system call returns error code 0BH if the BDOS cannot locate the FCB's directory entry when attempting to verify that the referenced FCB contains current information. The system call only returns this error for files open in Unlocked mode.

The system call returns error code 0FFH if a physical error is encountered and the BDOS Error mode is in one of the return modes (refer to the F\_ERRMODE system call). If the Error mode is in the default mode, the file system displays a message at the console identifying the physical error and terminates the calling process. When a physical error is returned to the calling process, it is identified by the four low-order bits of register AH as shown below:

01H - Disk I/O Error : permanent error

04H - Invalid Drive : drive select error

On all error returns except for physical error returns, AL = 255, F\_READRAND sets register AH to the number of records successfully read before the error was encountered. This value can range from 0 to 127 depending on the current BDOS Multisector Count. It is always set to zero when the Multisector Count is equal to one.

F_RENAME	
Rename A Disk File	
Entry Parameters:	
Register	CL: 017H (23)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	AH: Physical or Extended Error
	BX: Same as AX

The F\_RENAME system call uses the referenced FCB to change all directory entries of the file specified by the drive and filename in bytes 0 to 11 of the FCB to the filename specified in bytes 17 through 27.

If the file specified by the first filename is password-protected, the correct password must be placed in the first eight bytes of the current DMA buffer, or have been previously established as the default password (refer to the F\_PASSWD system call).

The calling process must also ensure that the filenames specified in the FCB are valid and unambiguous, and that the new filename does not already exist on the drive. F\_RENAME uses the drive code at byte 0 of the FCB to select the drive. The drive code at byte 16 of the FCB is ignored.

Interface attribute F5' specifies whether an extended file lock is to be maintained after the F\_ATTRIB call as shown below:

F5' = 0 - Do not maintain an extended file lock (default)

F5' = 1 - Maintain an extended file lock

If F5' is set and the referenced FCB specifies a file with an extended file lock, the calling process maintains the lock on the file. Otherwise, the file becomes available to other processes on the system. Section 2.11 describes extended file locking in detail.

A process can rename a file that it has open if the file is open in Locked mode. However, the BDOS returns a checksum error if the process subsequently references the file with a system call requiring an open FCB. A file open in Read-Only or Unlocked mode cannot be renamed by any process.

Renaming an open file can adversely affect the performance of the calling process. For this reason, you should close an open file before you rename it.

Upon return, the F\_RENAME system call returns a directory code in register AL with the value 00H if the rename is successful, or 0FFH if the file named by the first filename in the FCB is not found. Register AH is set to 00H in both of these cases. If a physical or extended error is encountered, the F\_RENAME system call performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error, and terminates the process. Otherwise, it returns to the calling process with register AL set to 0FFH and with register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 03H - Read/Only File
- 04H - Invalid Drive : drive select error
- 05H - File open by another process
- 07H - Password Error
- 08H - File Already Exists
- 09H - Illegal ? in FCB

F_SFIRST	
Find The First File That Matches The Specified FCB	
Entry Parameters:	
Register	CL: 011H (17)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	AH: Physical or Extended Error
	BX: Same as AX

The F\_SFIRST system call scans the directory for a match with the referenced FCB. Two types of searches can be performed. For standard searches, the calling process initializes bytes 0 through 12 of the referenced FCB, with byte 0 specifying the drive directory to be searched, bytes 1 through 11 specifying the file or files to be searched for, and byte 12 specifying the extent. Byte 12 is usually set to 00H. An ASCII question mark (63, or 03FH hexadecimal) in any of the bytes 1 through 12 matches all entries on the directory in the corresponding position. This facility, called ambiguous file reference, can be used to search for multiple files on the directory. When called in the standard mode, F\_SFIRST scans for the first file entry in the specified directory that matches the FCB and belongs to the current user number.

The F\_SFIRST system call also initializes the F\_SNEXT system call. After the search system call has located the first directory entry matching the referenced FCB, F\_SNEXT can be called repeatedly to locate all remaining matching entries. In terms of execution sequence, however, the F\_SNEXT call must follow either a F\_SFIRST or F\_SNEXT call with no other intervening BDOS file-access system calls.

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

Upon return, the F\_SFIRST system call returns a directory code in register AL with the value 0 to 3 if the search is successful, or 0FFH if a matching directory entry is not found. Register AH is set to zero in both of these cases. For successful searches, the current DMA is also filled with the directory record containing the matching entry, and the relative starting position is  $AL * 32$ . The directory information can be extracted from the buffer at this position.

If the directory has been initialized for date and time stamping, then an FCB resides in every fourth directory entry, and successful directory codes are restricted to the values 0 to 2. For successful searches, if the matching directory record is an extent zero entry, and if an SFCB resides at offset 96 within the current DMA buffer, then the contents of  $(DMA\ Address + 96) = 021H$ , and the SFCB contains the time and date stamp information and password mode for the file. This information is located at the relative starting position of  $97 + (AL * 10)$  within the current DMA in the following format:

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

Refer to Section 2.8 for more information about SFCBs.

If a physical error is encountered, the F\_SFIRST system call performs different actions depending on the BDOS error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message identifying the error at the console and terminates the calling process. Otherwise, it returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical error codes:

- 01H - Disk I/O Error : permanent error
- 04H - Invalid Drive : drive select error

F_SIZE	
Compute The Size Of A Disk File	
Entry Parameters:	
Register	CL: 023H (35)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Directory Code
	AH: Physical or Extended Error
	BX: Same as AX
	Random Record Field of FCB Set

The F\_SIZE system call determines the virtual file size. This is the address of the record immediately following the end of the file. The virtual size of a file corresponds to the physical size if the file is written sequentially. If the file is written in random mode, gaps might exist in the allocation, and the file might contain fewer records than the indicated size. For example, if a single record with record number 262,143, the Concurrent CP/M-86 maximum, is written to a file using the F\_WRITERAND system call, then the virtual size of the file is 262,144 records even though only one data block is actually allocated.

To compute file size, the calling process passes the address of an FCB with bytes R0, R1, and R2 present. The F\_SIZE system call sets the random record field of the FCB to the Random Record Number + 1 of the last record in the file. If the R2 byte is set to 04H, and R0 and R1 are both zero, then the file contains the maximum record count, 262,144.



A process can append data to the end of an existing file by calling F\_SIZE to set the random record position to the end of file, and then performing a sequence of random writes.

**Note:** the file need not be open in order to use F\_SIZE. However, if the file is open in Locked mode and it has been extended by the calling process, the file must be closed before F\_SIZE is called. Otherwise, F\_SIZE returns an incorrect file size. F\_SIZE returns the correct size for files open in Unlocked mode and Read-Only mode.

Upon return, F\_SIZE returns a 00H in register AL if the file specified by the referenced FCB is found, or a 0FFH in register AL if the file is not found. Register AH is set to 00H in both cases.

If a physical or extended error is encountered, F\_SIZE performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error and terminates the process. Otherwise, F\_SIZE returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 04H - Invalid Drive : drive select error
- 09H - Illegal ? in FCB

F_SNEXT	
Find A Subsequent File That Matches The Specified FCB Of A Previous F_SFIRST Or F_SNEXT	
Entry Parameters:	
Register CL:	012H (18)
Returned Values:	
Register AL:	Directory Code
AH:	Physical or Extended Error
BX:	Same as AX

The F\_SNEXT system call is identical to F\_SFIRST except that the directory scan continues from the last entry that was matched. F\_SNEXT returns a directory code in register AL, analogous to F\_SFIRST.

**Note:** in execution sequence, a F\_SNEXT call must follow either a F\_SFIRST or another F\_SNEXT with no other intervening BDOS file-access system calls.

F_TIMEDATE	
Return File Date Stamps And Password Mode	
<p>Entry Parameters:</p> <p>Register CL: 066H (102)</p> <p>DX: FCB Address - Offset</p> <p>DS: FCB Address - Segment</p> <p>Returned Values:</p> <p>Register AL: Directory Code</p> <p>AH: Physical Error</p> <p>BX: Same as AX</p>	

The F\_TIMEDATE system call returns the time and date stamp information and password mode for the specified file in byte 12 and bytes 24 through 31 of the specified FCB. The calling process passes the address of an FCB in which the drive, filename, and type fields have been defined.

If F\_TIMEDATE is successful, it sets the following fields in the referenced FCB

byte 12            password mode field

bit 7 - Read mode

bit 6 - Write mode

bit 5 - Delete mode

Byte 12 equal to 0 indicates the file has not been assigned a password.

byte 24 - 27    XFCB Create or Access time stamp field

byte 28 - 31    XFCB Update time stamp field

Upon return, F\_TIMEDATE returns a directory code in register AL with the value 00H if the operation is successful, or 0FFH if the specified file is not found. Register AH is set to 00H in both of these cases. If a physical or extended error is encountered, F\_TIMEDATE performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error and terminates the calling process. Otherwise, F\_TIMEDATE returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical error codes:

- 01H - Disk I/O Error : permanent error
- 04H - Invalid Drive : drive select error
- 09H - Illegal ? in FCB

F_TRUNCATE	
Truncate File	
Entry Parameters:	
Register CL:	063H (99)
Register DX:	FCB Address - Offset
Returned Values:	
Register AL:	Directory Code
Register AH:	Physical or Extended Error
Register BX:	Same as AX

The F\_TRUNCATE system call sets the last record of a file to the Random Record Number contained in the referenced FCB. The calling program passes the address of the FCB in register DX with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and bytes 33 through 35 (R0, R1, and R2) specifying the last record of the file. The last record number is a 24-bit value, stored with the least significant byte first (R0), the middle byte next (R1), and the high byte last (R2). This value can range from 0 to 262,143 (03FFFFH).

If the file specified by the referenced FCB is password-protected, the correct password must have been placed in the first eight bytes of the current DMA buffer, or have been previously established as the default password (refer to the F\_PASSWD system call).

Interface attribute F5' specifies whether an extended file lock is to be maintained after the F\_TRUNCATE call, as shown below:

F5' = 0 - Do not maintain an extended file lock (default)

F5' = 1 - Maintain an extended file lock

If F5' is set and the referenced FCB specifies a file with an extended file lock, the calling process maintains the lock on the file. Otherwise, the file becomes available to other processes on the system. Section 2.11 describes extended file locking in detail.

F\_TRUNCATE requires that the Random Record Number field of the referenced FCB specify a value less than the current file size. In addition, if the file is sparse, the random record field must specify a region of the file where data exists.

A process can truncate a file that it currently has open if the file is opened in locked mode, and the file has not been extended during the open session. However, the BDOS returns a checksum error if the process makes a subsequent reference to the file with a BDOS system call requiring an open FCB. A process cannot truncate files open in R/O or Unlocked mode.

Truncating an open file is not recommended under Concurrent CP/M-86. F\_TRUNCATE truncates a file based on the file's state in the directory. If a process attempts to truncate at a region of the file that has been allocated in memory but has not been recorded in the directory, F\_TRUNCATE returns an error. Even when successful, an open file truncate can adversely affect the performance of the calling process. For these reasons, you should close an open file before you truncate it.

After completion, F\_TRUNCATE returns a directory code in register AL with the value 00H if the operation is successful or 0FFH if the file is not found or if the record number is invalid. In both cases register AH is set to 00H.

If a physical or extended error is encountered, F\_TRUNCATE performs different actions depending on the BDOS Error mode (refer to F\_ERRMODE). If the BDOS Error mode is in the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, F\_TRUNCATE returns to the calling program with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 03H - Read/Only File
- 04H - Invalid Drive : drive select error
- 05H - File Currently Open
- 06H - Close Checksum Error
- 07H - Password Error
- 08H - File Already Exists
- 09H - Illegal ? in FCB
- 0AH - Open File Limit Exceeded
- 0BH - No Room in System Lock List

F_UNLOCK	
Unlock Records In A Disk File	
Entry Parameters:	
Register	CL: 02BH (43)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Error Code
	AH: Physical Error
	BX: Same as AX

The F\_UNLOCK system call unlocks one or more consecutive records previously locked by the F\_LOCK system call. This system call is only supported for files open in Unlocked mode. If it is called for a file open in Locked or Read-Only mode, no unlocking action occurs and a successful result is returned. Record locking and unlocking is described in detail in Section 2.14.

The calling process passes the address of an FCB in which the Random Record Field is filled with the Random Record Number of the first record to be unlocked. The number of records to be unlocked is determined by the BDOS Multisector Count (refer to the F\_MULTISEC system call). The current DMA must contain the 2-byte File ID returned by the F\_OPEN system call when the referenced FCB was opened. Note that the File ID is only returned by F\_OPEN when the file open mode is Unlocked.



If interface attribute F5' is set to 1, F\_UNLOCK unlocks all locked records belonging to the calling process. The F\_UNLOCK interface attribute definition is listed below:

- F5' = 0 - Unlock records specified by Random Record Number and BDOS Multisector Count (default)
- F5' = 1 - Unlock all locked records.

F\_UNLOCK ignores the FCB Random Record field and the BDOS Multisector Count when F5' is set.

F\_UNLOCK does not unlock a record that is currently locked by another process. However, the system call does not return an error if a process attempts to do that. Thus, if the Multisector Count is greater than one, F\_UNLOCK unlocks all records locked by the calling process, skipping those records locked by other processes.

Some F\_UNLOCK requests require a new entry in the BDOS system Lock List. If there is insufficient space in the system Lock List to satisfy the F\_UNLOCK request, or if the process record Lock List limit is exceeded, then F\_UNLOCK does not unlock any records and returns an error code to the calling process.

Upon return, F\_UNLOCK sets register AL to 00H if the unlock operation was successful. Otherwise, register AL contains one of the following error codes:

- 01H - Reading unwritten data
- 03H - Cannot close current extent
- 04H - Seek to unwritten extent
- 06H - Random Record Number out of range
- 0AH - FCB Checksum Error
- 0CH - Process record Lock List limit exceeded
- 0DH - Invalid File ID
- 0EH - No room in system Lock List
- 0FFH - Physical error refer to register AH

The system call returns error code 01H when it accesses a data block which has not been previously written.

The system call returns error code 03H when it cannot close the current extent prior to moving to a new extent.

The system call returns error code 04H when it accesses an extent that has not been created.

The system call returns error code 06H when byte 35 (r2) of the referenced FCB is greater than 3.

The system call returns error code 0AH if the referenced FCB failed the FCB checksum test.

The system call returns error code 0CH if performing the unlock request would require that the process consume more than the maximum allowed number of system Lock List entries.

The system call returns error code 0DH when an invalid File ID is placed at the beginning of the current DMA.

The system call returns error code 0EH when the system Lock List is full and performing the unlock request would require at least one new entry.

The system call returns error code 0FFH if a physical error was encountered and the BDOS Error mode is one of the return modes (refer to the F\_ERRMODE system call). If the Error mode is the default mode, the system displays a message at the console identifying the physical error and terminates the calling process. When the system call returns a physical error to the calling process, it is identified by register AH as shown below:

01H - Disk I/O Error : permanent error

04H - Invalid Drive : drive select error

F_USERNUM	
Set Or Return The Calling Process's Default User Number	
Entry Parameters:	
Register	CL: 020H (32)
	DL: 0FFH to GET User Number User Number to SET
Returned Values:	
Register	AL: Current User Number if GET
	BL: Same as AL

A process can change or interrogate its current default user number by calling F\_USERNUM. If register DL = 0FFH, then the system call returns the value of this user number in register AL. The value can range from 0 to 0FH. If register DL is not 0FFH, then the system call changes the default user number to the value in DL, module 010H (the high nibble of DL is masked off).

Under Concurrent CP/M-86, a new process inherits its initial default user number from its parent, the process creating the new process. Changing the default user number does not change the user code of the parent. On the other hand, all child processes of the calling process inherit the new user number.

This convention is demonstrated by the operation of the TMP. When a command is typed, a new process is created with the same user number as that of the TMP. If this new process changes its user number, the TMP is unaffected. Once the new process terminates, the TMP displays the same user number in its prompt that it displayed before the command was entered and the child process was created.

F_WRITE	
Write Records Sequentially To A Disk File	
Entry Parameters:	
Register	CL: 015H (21)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Error Code
	AH: Physical Error
	BX: Same as AX

The F\_WRITE system call writes 1 to 128, 128-byte data records beginning at the current DMA address into the file named by the specified FCB. The BDOS Multisector Count (refer to the F\_MULTISEC system call) determines the number of 128-byte records that are written. The default is one record. An F\_OPEN or F\_MAKE system call must have previously activated the referenced FCB.

F\_WRITE places the record into the file at the position indicated by the CR byte of the FCB, and then automatically increments the CR byte to the next record position. If the CR field overflows, the system call automatically opens or creates the next logical extent and resets the CR field to 00H in preparation for the next write operation. If F\_WRITE is used to write to an existing file, then the newly written records overlay those already existing in the file. The calling process must set the CR field to 00H following an F\_OPEN or F\_MAKE system call if the intent is to write sequentially from the beginning of the file.

F\_WRITE makes an update date and time stamp for the file if the following conditions are met: the referenced drive has a directory label that requests update date and time stamping, and the file has not already been stamped for update by a previous F\_MAKE or F\_WRITE system call.

Upon return, the F\_WRITE system call sets register AL to 00H if the write operation is successful. Otherwise, register AL contains an error code identifying the error as shown below:

- 01H - No available directory space
- 02H - No available data block
- 08H - Record locked by another process
- 09H - Invalid FCB
- 0AH - FCB Checksum Error
- 0BH - Unlocked file verification error
- 0FFH - Physical error; refer to register AH

The system call returns error code 01H when it attempts to create a new extent that requires a new directory entry, and no available directory entries exist on the selected disk drive.

The system call returns error code 02H when it attempts to allocate a new data block to the file, and no unallocated data blocks exist on the selected disk drive.

The system call returns error code 08H if the calling process attempts to write to a record locked by another process, or a record locked by the calling process in shared mode. The system call returns this error only for files open in Unlocked mode.

The system call returns error code 09H if the FCB is invalidated by a previous F\_CLOSE system call that returned an error.

The system call returns error code 0AH if the referenced FCB failed the FCB checksum test.

The system call returns error code 0BH if the BDOS cannot locate the FCB's directory entry when attempting to verify that the referenced FCB contains current information. The system call returns this error only for files open in Unlocked mode.

The system call returns error code 0FFH if a physical error was encountered and the BDOS is in Return Error mode or Return and Display Error mode (refer to the F\_ERRMODE system call). If the Error mode is the default mode, the system displays a message at the console identifying the physical error and terminates the calling process. When the system call returns a physical error to the calling process, it is identified by register AH as shown below:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 03H - Read/Only File or  
File Opened in Read/Only Mode or  
File password protected in Write mode
- 04H - Invalid Drive : drive select error

On all error returns, except for physical error returns (AL = 255), F\_WRITE sets register AH to the number of records successfully read before the error was encountered. This value can range from 0 to 127, depending on the current BDOS Multisector Count. It is always set to zero when the Multisector Count is equal to one.

F_WRITERAND	
Write Random Records To A Disk File	
Entry Parameters:	
Register	CL: 022H (34)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Error Code
	AH: Physical Error
	BX: Same as AX

The F\_WRITERAND system call is analogous to the F\_READRAND system call, except that data is written to the disk from the current DMA address. If the disk extent and/or data block where the data is to be written is not already allocated, the BDOS automatically performs the allocation before the write operation continues.

In order to write to a file using the F\_WRITERAND system call, the calling process must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. If the file is empty, the calling process must create the base extent with the F\_MAKE system call before an F\_WRITERAND system call. The base extent might or might not contain data, but it records the file in the directory so that it can be displayed by the DIR utility. If a process does not open extent 0 and allocates data to some other extent, the file is invisible to the DIR utility.

The F\_WRITERAND system call sets the logical extent and current record positions to correspond with the random record being written, but does not change the Random Record Number. Thus sequential read or write operations can follow a random write, with the current record being reread or rewritten as the calling process switches from random to sequential mode.

**F\_WRITERAND** makes an update date and time stamp for the file if the following conditions are met: the referenced drive has a directory label that requests update date and time stamping, and the file has not already been stamped for update by a previous **F\_MAKE** or **F\_WRITE** system call.

If the BDOS Multisector Count is greater than one (refer to the **F\_MULTISEC** system call), the **F\_WRITERAND** system call reads multiple consecutive records into memory beginning at the current DMA address. The system call automatically increments the R0, R1, and R2 field of the FCB to write each record. However, it restores the FCB's Random Record Number to the first record's value upon return to the calling process.

Upon return, the **F\_WRITERAND** system call sets register AL to 00H if the write operation is successful. Otherwise, register AL contains one of the following error codes:

- 02H - No available data block
- 03H - Cannot close current extent
- 05H - No available directory space
- 06H - Random record number out of range
- 08H - Record locked by another process
- 0AH - FCB Checksum Error
- 0BH - Unlocked file verification error
- 0FFH - Physical error refer to register AH

The system call returns error code 02H when it attempts to allocate a new data block to the file. No unallocated data blocks exist on the selected disk drive.

The system call returns error code 03H when it cannot close the current extent before moving to a new extent.

The system call returns error code 05H when it attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.



The system call returns error code 06H when byte 35 (R2) of the referenced FCB is greater than 3.

The system call returns error code 08H if the calling process attempts to write to a record locked by another process, or a record locked by the calling process in shared mode. The system call returns this error only for files open in Unlocked mode.

The system call returns error code 0AH if the referenced FCB failed the FCB checksum test.

The system call returns error code 0BH if the BDOS cannot locate the FCB's directory entry when attempting to verify that the referenced FCB contains current information. The system call returns this error only for files open in Unlocked mode.

The system call returns error code 0FFH if a physical error is encountered and the BDOS Error mode is in one of the return modes (refer to the F\_ERRMODE system call). If the Error mode is in the default mode, the system displays a message at the console identifying the physical error and terminates the calling process. When a physical error is returned to the calling process, it is identified by register AH as shown below:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 03H - Read/Only File or  
File Opened in Read/Only Mode or  
File password protected in Write mode
- 04H - Invalid Drive : drive select error

On all error returns, except for physical error returns (AL = 255), F\_WRITERAND sets register AH to the number of records successfully read before the error was encountered. This value can range from 0 to 127 depending on the current BDOS Multisector Count. It is always set to zero when the Multisector Count is equal to one.

F_WRITEXFCB	
Write Extended File Control Block Of A Disk File	
Entry Parameters:	
Register CL:	067H (103)
DX:	FCB Address - Offset
DS:	FCB Address - Segment
Returned Values:	
Register AL:	Directory Code
AH:	Physical or Extended Error
BX:	Same as AX

The F\_WRITEXFCB system call creates a new XFCB or updates the existing XFCB for the specified file. The calling process passes the address of an FCB in which the drive, name, type, and extent fields have been defined. The FCB extent field, if set, specifies the password mode and whether a new password is to be assigned to the file. The format of the extent field byte is shown below:

FCB byte 12 (EX)    XFCB password mode

bit 7 - Read mode

bit 6 - Write mode

bit 5 - Delete mode

bit 0 - assign new password to the file

If the FCB is currently password-protected, the correct password must reside in the first 8 bytes of the current DMA or have been previously established as the default password (refer to the F\_PASSWD system call). If bit 0 is set to 1, the new password must reside in the second 8 bytes of the current DMA.

**Note:** the F\_WRITEXFCB system call does not create or update an XFCB if the XFCB specifies a file open by another process. However, a process can update or create an XFCB for a file that it has open in Locked mode.

Upon return, F\_WRITEXFCB returns a directory code in register AL with the value 00H if the XFCB create or update was successful. F\_WRITEXFCB returns 0FFH in register AL if no directory label existed on the specified drive, or the file specified in the FCB was not found, or no space existed in the directory to create an XFCB, or if the drive is not password enabled. F\_WRITEXFCB also returns 0FFH if passwords are not enabled by the specified drive's directory label. Register AH is set to 00H in all of these cases.

If a physical or extended error is encountered, F\_WRITEXFCB performs different actions depending on the BDOS Error mode (refer to the F\_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error and terminates the calling process. Otherwise, F\_WRITEXFCB returns to the calling process with register AL set to 0FFH and register AH set to one of the following physical or extended error codes:

- 01H - Disk I/O Error : permanent error
- 02H - Read/Only Disk
- 04H - Invalid Drive : drive select error
- 05H - File open by another process,  
or open in Read-Only or Unlocked mode
- 07H - Password Error
- 09H - Illegal ? in FCB

F_WRITEZF	
Write A Random Record To A Disk File And Prefill New Data Blocks With Zeros	
Entry Parameters:	
Register	CL: 028H (40)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AL: Error Code
	AH: Physical Error
	BX: Same as AX

The F\_WRITEZF system call is similar to the F\_WRITERAND system call, with the exception that it fills a previously unallocated data block with zeros (00H) before writing the record. If this system call has been used to create a file, records accessed by an F\_READRAND system call that contain all zeros identify unwritten Random Records. Unwritten random records in allocated data blocks of files created using the F\_WRITERAND system call contain uninitialized data.

### 6.2.5 List Device I/O System Calls

L_ATTACH
Attach To The Calling Process's Default List Device
Entry Parameters: Register CL: 09EH (158)

The L\_ATTACH system call attaches the default list device of the calling process. If the list device is already attached to some other process, the calling process relinquishes the CPU until the other process detaches from the list device. When the list device becomes free, and the calling process is the highest priority process waiting for the list device, the attach operation occurs.

Refer to Table 6-5 for a list of error codes returned in CX.

L_CATTACH	
Conditionally Attach To The Default List Device	
Entry Parameters:	
Register CL:	0A1H (161)
Returned Values:	
Register AX:	0 if attach 'OK' 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The L\_CATTACH system call attaches the default list device of the calling process only if the list device is currently available.

If the list device is currently attached to another process, the system call returns a value of 0FFH, indicating that the list device could not be attached. The system call returns a value of 00H to indicate that either the list device is already attached to the process, or that it was unattached, and a successful attach operation was made.

Refer to Table 6-5 for a list of error codes returned in CX.

L_DETACH	
Detach The Calling Process's Default List Device	
Entry Parameters:	
Register CL:	09FH (159)
Returned Values:	
Register AX:	0 if detach 'OK' 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The L\_DETACH system call detaches the default list device of the calling process. If the list device is not currently attached, no action takes place.

Refer to Table 6-5 for a list of error codes returned in CX.

L_GET	
Return The Calling Process's Default List Device	
Entry Parameters:	
Register CL:	0A4H (164)
Returned Values:	
Register AL:	List Device Number
Register BL:	Same as AL

The L\_GET system call returns the default list device number of the calling process.



L_SET	
Set The Calling Process's Default List Device	
Entry Parameters:	
Register	CL: 0A0H (160) DL: List Device
Returned Values:	
Register	CX: Error Code

The L\_SET system call sets the default list device for the calling process.

Refer to Table 6-5 for a list of error codes returned in CX.

L_WRITE	
Write A Character To The Default List Device	
Entry Parameters:	
Register	CL: 05H (5)
	DL: Character

The L\_WRITE system call writes the specified character to the default list device of the calling process. Before writing the character, the system internally calls L\_ATTACH to verify that the calling process owns its default list device.

L_WRITEBLK *	
Send Specified Character String To Default List Device	
Entry Parameters:	
Register CL:	070H (112)
Register DX:	CHCB Address

L\_WRITEBLK sends the character string specified in the Character Control Block (CHCB) and addressed in register pair DX to the logical list device, LST:. The CHCB format is

bytes 0 - 1 : Offset of character string  
 bytes 2 - 3 : Segment of character string  
 bytes 4 - 5 : Length of character string to print

### 6.2.6 Memory System Calls

There are two classes of Memory system calls in Concurrent CP/M-86. The first class supports the MP/M-86 memory allocation scheme and contains two system calls, M\_ALLOC and M\_FREE. The second class contains six system calls, MC\_ABSMAX, MC\_ALLFREE, MC\_ALLOC, MC\_ABSALLOC, MC\_FREE, and MC\_MAX. These system calls support the CP/M-86 memory allocation scheme.

**Note:** the CP/M-86 memory calls are also supported under MP/M-86.

Many of the Memory system calls use the Memory Control Block (MCB) or the Memory Parameter Block (MPB) to pass parameters to and from the operating system. The format, structure and example programming equates for these data structures are presented below, along with example listings.



Figure 6-7. MCB - Memory Control Block

Table 6-13. MCB Field Definitions

<i>Field</i>	<i>Definition</i>
BASE	The Segment Address of the beginning of the specified memory segment.
LENGTH	Length of the Memory Segment in paragraphs. The LENGTH field is set to the number of paragraphs wanted.
EXT	The EXT field is unused but must be available.

```

; *****
;
; *
; *      Memory Control Block Definition
; *
; *
; *****

```

```

mcb_base      equ      word ptr 0
mcb_length    equ      word ptr mcb_base + word
mcb_ext       equ      byte ptr mcb_length + word

mcb_len       equ      mcb_ext + byte

```

Listing 6-1. Memory Control Block Definition

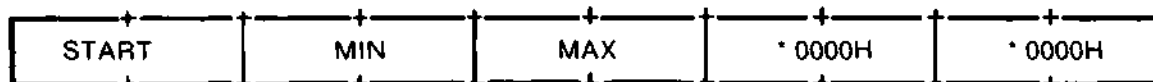


Figure 6-8. MPB - Memory Parameter Block

Table 6-14. MPB Field Definitions

<i>Field</i>	<i>Description</i>
START	if non-00H, an absolute request at this paragraph
MIN	minimum memory needed (paragraphs)
MAX	maximum memory wanted (paragraphs)
* 0000H	these fields must be 00H; they are used internally.

```

; *****
; *
; *      Memory Parameter Block Definition
; *
; *****

```

```

mpb_start      equ      word ptr 0
mpb_min        equ      word ptr mpb_start + word
mpb_max        equ      word ptr mpb_min + word
mpb_pdadr      equ      word ptr mpb_max + word
mpb_flags      equ      word ptr mpb_pdadr + word

mpb_len        equ      mpb_flags + word

```

```

; mpb_flags definition

```

```

mf_load        equ      00001h
mf_share       equ      00002h
mf_code        equ      00004h

```

```

;

```

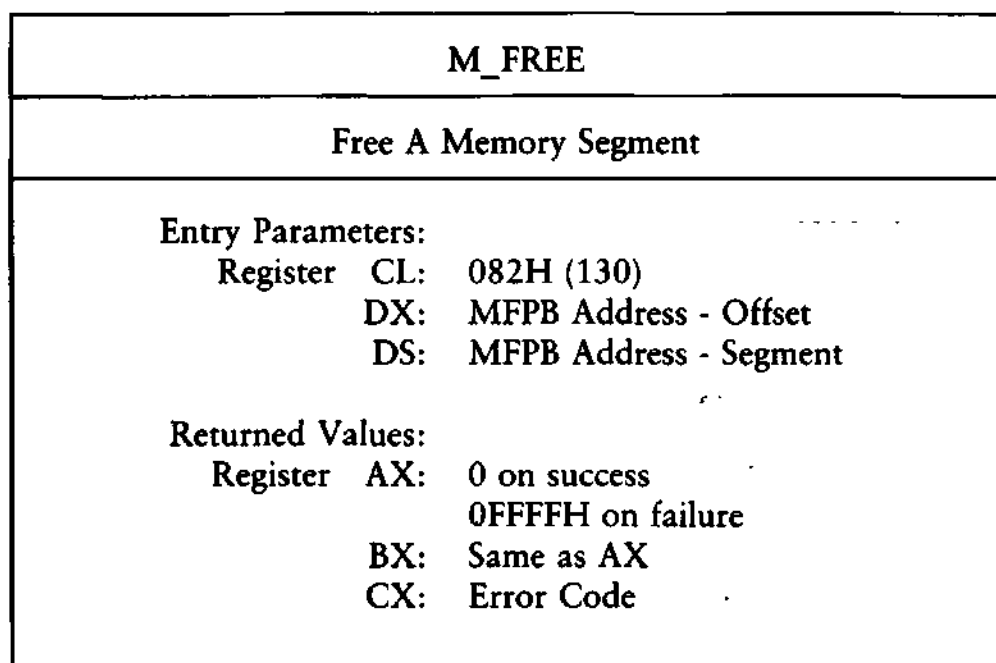
Listing 6-2. Memory Parameter Block Definition

M_ALLOC	
Allocate A Memory Segment	
Entry Parameters:	
Register	CL: 080H or 081H (128,129)
	DX: MPB Address-Offset
	DS: MPB Address-Segment
	MPB filled in
Returned Values:	
Register	AX: 0 on success
	0FFFFH on failure
	BX: Same as AX
	CX: Error Code
	MPB_start filled in

The M\_ALLOC system call allows a program to allocate extra memory. A successful allocation allocates a contiguous memory segment whose length is at least the MIN and no more than the MAX number of paragraphs specified in the MPB. The START field of the MPB is modified to be the starting paragraph of the memory segment. The MIN and MAX fields are modified to be the length of the memory segment in paragraphs. Memory Segments can be explicitly released through the M\_FREE system call; Concurrent CP/M-86 also releases all memory owned by a process at termination.

**Note:** MIN and MAX fields must be explicitly filled in. The MAX value must be greater than or equal to the MIN value.

Refer to Table 6-5 for a list of error codes returned in CX.



**Figure 6-9. MFPB - M\_FREE Parameter Block**

The M\_FREE system call releases memory starting at the START paragraph to the end of a single previously allocated segment that contains the START paragraph. If the START paragraph is the same as that returned in the MPB of a memory allocation call, then M\_FREE releases the whole memory segment. The \* 0000H field must be initialized to zero.

Refer to Table 6-5 for a list of error codes returned in CX.

MC_ABSALLOC	
Allocate A Memory Segment At A Specified Address	
Entry Parameters:	
Register	CL: 038H (56)
	DX: MCB Address - Offset
	DS: MCB Address - Segment
Returned Values:	
Register	AL: 0 on success
	0FFH on failure
	BL: Same as AL
	CX: Error Code

The MC\_ABSALLOC system call allocates a memory area that starts at the address specified by the BASE field. The memory area's length is specified by the LENGTH field of the MCB. Upon return, register AL contains a 00H if the request was successful, and a 0FFH if the memory could not be allocated. If the calling process already owns the requested memory, no error is returned. This assures compatibility with CP/M-86.

Refer to Table 6-5 for a list of error codes returned in CX.



MC_ABSMAX	
Allocate Maximum Memory Available At A Specified Address	
Entry Parameters:	
Register	CL: 036H (54)
	DX: MCB Address - Offset
	DS: MCB Address - Segment
	MCB_base filled in, MCB_length set to max number of paragraphs wanted
Returned Values:	
Register	AL: 0 on success
	0FFH on failure
	BL: Same as AL
	CX: Error Code
	MCB_length set to actual number of paragraphs allocated

In CP/M-86, system call 036H does not allocate memory, but under Concurrent CP/M-86, this system call allocates memory, because other processes are competing for common memory. For compatibility with CP/M-86, MC\_ABSALLOC (system call 56) does not return an error if there is a memory segment allocated at the absolute address.

MC\_ABSMAX is used to allocate the largest possible region at the absolute paragraph boundary given by the BASE field of the MCB, for a maximum of LENGTH paragraphs. If the allocation is successful, the system call sets the LENGTH to the actual length. Upon return, register AL has the value 0FFH if no memory is available at the absolute address, and 00H if the request was successful.

Refer to Table 6-5 for a list of error codes returned in CX.

MC_ALLFREE
Free All Memory Owned By The Calling Process
Entry Parameters: Register CL: 03AH: (58)

In the Concurrent CP/M-86 environment, the MC\_ALLFREE system call releases all of the calling process's memory except the User Data Area (UDA). This system call is useful for system processes and for subprocesses that share the memory of another process.

**Note:** this system call should not be used by processes running programs loaded into the Transient Program Areas (TPAs).

MC_ALLOC	
Allocate A Memory Segment	
Entry Parameters:	
Register	CL: 037H (55)
	DX: MCB Address - Offset
	DS: MCB Address - Segment
	MCB_length filled in
Returned Values:	
Register	AL: 0 on success
	0FFH on failure
	BL: Same as AL
	CX: Error Code
	MCB_base filled in

The MC\_ALLOC system call allocates a memory area whose size is the LENGTH field of the MCB. MC\_ALLOC returns the base paragraph address of the allocated region in the user's MCB. Upon return, register AL contains a 00H if the request was successful, and a 0FFH if the memory could not be allocated.

Refer to Table 6-5 for a list of error codes returned in CX.

MC_FREE	
Free A Specified Memory Segment	
Entry Parameters:	
Register	CL: 039H (57)
	DX: MCB Address - Offset
	DS: MCB Address - Segment
	MCB_base, MCB_ext filled in
Returned Values:	
Register	AL: 0 if successful
	0FFH on failure
	BL: Same as AL
	CX: Error Code

The MC\_FREE system call is used to release memory areas allocated to the program. The value of the EXT field of the MCB controls the operation of this system call. If EXT = 0FFH, then the system call releases all memory areas allocated by the calling program. If the EXT field is 00H, the system call releases the memory area beginning at the specified BASE and ending at the end of the previously allocated memory segment.

Refer to Table 6-5 for a list of error codes returned in CX.

MC_MAX	
Allocate Maximum Memory Available	
<p>Entry Parameters:</p> <p>Register CL: 035H (53)</p> <p>DX: MCB Address - Offset</p> <p>DS: MCB Address - Segment</p> <p>(MCB_length contains maximum number of paragraphs wanted)</p> <p>Returned Values:</p> <p>Register AL: 0 on success 0FFH on failure</p> <p>BL: Same as AL</p> <p>CX: Error Code</p> <p>(MCB_base filled in, MCB_length set to actual number of paragraphs allocated)</p>	

In CP/M-86, system call 035H does not allocate memory, but under Concurrent CP/M-86, this system call allocates memory because other processes are competing for common memory. For compatibility with CP/M-86, MC\_ABSALLOC (system call 56) does not return an error if there is a memory segment allocated at the absolute address.

MC\_MAX allocates the largest available memory region that is less than or equal to the LENGTH field of the MCB in paragraphs. If the allocation is successful, the system call sets the BASE to the base paragraph address of the available area and LENGTH to the paragraph length. Upon return, register AL has the value 0FFH if no memory is available, and 00H if the request was successful. The system call sets the EXT to 1 if there is additional memory for allocation, and 0 if no additional memory is available.

Refer to Table 6-5 for a list of error codes returned in CX.

## 6.2.7 Process/Program System Calls

P_ABORT	
Terminate A Process By Name Or PD Address	
Entry Parameters:	
Register CL:	09DH (157)
DX:	APB Address - Offset
DS:	APB Address - Segment
	APB filled in
Returned Values:	
Register AX:	0 on success 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

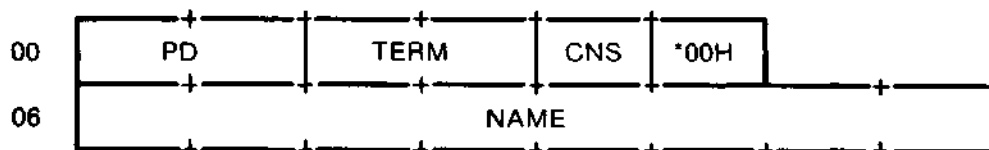


Figure 6-10. APB - Abort Parameter Block

Table 6-15. APB Field Definitions

<i>Field</i>	<i>Definition</i>
PD	Process Descriptor offset of the process to be terminated. If this field is zero, a match is attempted with the NAME and CNS fields to find the process. If this field is nonzero, the NAME and CNS fields are ignored.
TERM	Termination Code. This field corresponds to the termination code of the P_TERM system call. If the low-order byte of TERM is 0FFH, P_ABORT can abort a specified system process; if the termination code is not 0FFH, the system call can only terminate a user process. (A system process is identified by the SYS flag in the Process Descriptor's FLAG field.)
*00H	This field is reserved for future use and must be set to zero.
CNS	Default console of process to be aborted. If the PD field is 0, the P_ABORT system call scans the Thread List for a PD with the same NAME and CNS fields as specified in the APB. P_ABORT only aborts the first process that it finds. Subsequent calls must be made to abort all processes with the same NAME and CNS.
NAME	Name of the process to be aborted. Combined with the CNS field, the NAME field is used to find the process to be aborted. This is only used if the PD field is 0.

The P\_ABORT system call permits a process to terminate another specified process. The calling process passes the address of a data structure called an Abort Parameter Block, initialized as described above.

If the Process Descriptor address is known, it can be filled in, and the process name and console can be omitted. Otherwise, the Process Descriptor address field should be a 00H and the process name and console must be specified. In either case, the calling process must supply the termination code, which is the same parameter passed to the P\_TERM system call.

Refer to Table 6-5 for a list of error codes returned in CX.



P_CHAIN	
Load, Initialize And Jump To Specified Program	
Entry Parameters:	
Register CL:	02FH (47)
DMA Buffer:	Command Line
Returned Values:	
Register AX:	0FFFFH - Could not find Command

The P\_CHAIN system call provides a means of chaining from one program to the next without operator intervention. Although there is no passed parameter for this call, the calling process must place a command line terminated by a 0 byte in the default DMA buffer.

Under Concurrent CP/M-86, the P\_CHAIN system call releases the memory of the calling process before executing the command. The command is processed in the same manner as the P\_CLI system call. If the command warrants the loading of a CMD file and the memory released is large enough for the new program, Concurrent CP/M-86 loads the new program into the same memory area as the old program. The new program is run by the same process that ran the old program. The name of the process is changed to reflect the new program being run.

Parameter passing between the old and new programs is accomplished through the use of disk files, queues, or the command line. The command line is parsed and placed in the Base Page of the new program in the manner documented in the P\_CLI system call.

The P\_CHAIN system call returns an error if no CMD file is found. If a CMD file is found, and an error occurs after it is successfully opened, the calling process terminates, as its memory has been released.

P_CLI	
Interpret And Execute Command Line	
Entry Parameters:	
Register CL:	096H (150)
DX:	CLBUF Address - Offset
DS:	CLBUF Address - Segment
Returned Values:	
Register AX:	0 on success 0FFFFH on error
CX:	Error Code

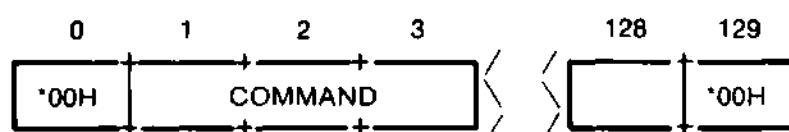


Figure 6-11. CLI Command Line Buffer

Table 6-16. Command Line Buffer Field Definitions

Field	Definition
*00H	Must be set to zero for internal use.
COMMAND	1-128 ASCII characters terminated with a null character.

The P\_CLI system call obtains an ASCII command from the Command Line Buffer (CLBUF) and then executes it. If the calling process is attached to its default virtual console, the P\_CLI system call assigns the virtual console to either the newly created process, or to the Resident System Process (RSP) that acts on the command. The calling process must reattach to its default virtual console before accessing it.

P\_CLI calls F\_PARSE to parse the command line. If an error occurs in F\_PARSE, P\_CLI returns to the calling process with the error code set to the same code that F\_PARSE returned.

If there is no disk specification for the command, P\_CLI tries to open a system queue with the same name as the command. If the open operation is successful, and the queue is an RSP-type queue, P\_CLI then writes the command tail to the RSP queue. If the queue is full, the system call returns an error code to the calling process. The P\_CLI function also attempts to assign the calling process's virtual console to a process with the same name as the RSP queue. If the RSP queue cannot be found, the CLI assumes the command is on disk and continues.

The P\_CLI system call opens a file with the filename being the command and the filetype being CMD. If the command has an explicit disk specification, and the F\_OPEN system call fails, P\_CLI returns an error code to the calling process. If there is no disk specification with the command, P\_CLI attempts to open the command file on the system disk. If the F\_OPEN system call succeeds, P\_CLI checks the file to verify the SYSTEM attribute is on. This search order is discussed in Section 2.9.1 of the *Concurrent CP/M-86 User's Guide*. If this second F\_OPEN fails or if the DIR attribute is on, P\_CLI returns an error code to the calling process.

Once the P\_CLI system call succeeds in opening the command file, it calls the P\_LOAD system call. The P\_LOAD system call finds, and then loads the file into an appropriate memory space. If P\_LOAD encounters any errors, the P\_CLI system call returns to the calling process with the error code set by the Load system call.

A successful load operation establishes the command file in memory with its Base Page partially initialized. The P\_CLI system call then continues parsing the command tail to set up the Base Page values from 050h to 0FFh.

P\_CLI initializes an unused Process Descriptor from the internal PD table, a UDA, and a 96-byte stack area. The UDA and stack are dynamically allocated from memory. P\_CLI then calls the P\_CREATE system call. If P\_CLI encounters an error in any of these steps, it releases all memory segments allocated for the new command, as well as the Process Descriptor, and then returns with the appropriate error code set.

Once the P\_CREATE system call returns successfully, the P\_CLI system call assigns the calling process's default virtual console to the new process and then returns.

The calling process should set its priority to less than the TMP (198) if it wants to attach to the virtual console after the created process releases it. Once the calling process has successfully reattached, it should set its priority back to 200.

Refer to Table 6-5 for a list of error codes returned in CX.

P_CREATE	
Create A Process	
Entry Parameters:	
Register CL:	090H (144)
DX:	PD Address - Offset
DS:	PD Address - Segment
	PD filled in
Returned Values:	
Register AX:	0 on success
	0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The P\_CREATE system call allows a process to create a subprocess within its own memory area. The child process shares all memory owned by the calling process at the time of the P\_CREATE call. If the Process Descriptor (PD) is outside of the operating system area, the system copies it into a PD from the internal PD Table. The system call returns an error code if there are no more unused PDs in the table.

The User Data Area (UDA) can be anywhere in memory but is required to be on a paragraph boundary. The only time the system copies the PD is if it is not within 64k of the System Data Segment.

Process Descriptors, as well as Queue Descriptors and Queue Buffers, are required to be within the System Data Segment because they are linked together on various system lists or are used by more than one process. Because of this, they cannot be in the Transient Process Area (TPA), where they cannot be protected.

More than one process can be created by a single P\_CREATE call if the LINK field of the PD is nonzero. In this case, it is assumed to point to another PD within the same Data Segment. After it creates the first process, the system call checks the Process Descriptor's LINK field. Using this linked list of PDs, a single P\_CREATE call can create multiple processes.

**Note:** the P\_CREATE system call does not check the validity of the PD addresses passed by the calling process. An invalid PD address can cause Concurrent CP/M-86 to crash if no hardware memory protection is available on the system.

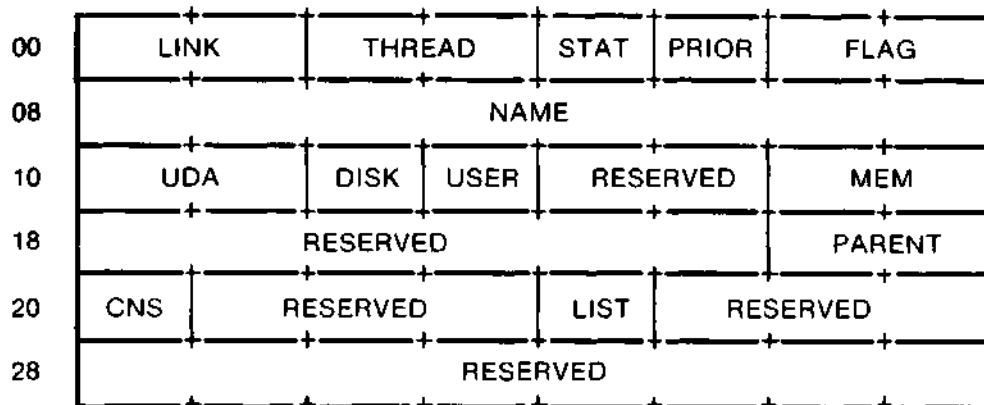


Figure 6-12. PD - Process Descriptor

Table 6-17. PD Field Definitions

<i>Field</i>	<i>Definition</i>										
LINK	Link field for insertion on current system list. If this field's initial value is nonzero, it is assumed to point to another PD. This field is used to create more than one process with a single Create Process call.										
THREAD	Link field for insertion on Thread List. Initialized to be zero (0).										
STAT	Current Process activity. Initialized to be zero (0). Activity codes are listed below: <table> <tr> <td>00 RUN</td><td>The process is ready to run. The STAT field is always in this state when a process is examining its own Process Descriptor. The PD is on the Ready List. The currently running process is always at the head of Ready List.</td></tr> <tr> <td>01 POLL</td><td>The process is polling a device. The PD is on the Poll List.</td></tr> <tr> <td>02 DELAY</td><td>The process is delaying for a specified number of system ticks. The PD is on the Delay List.</td></tr> <tr> <td>06 Read Queue</td><td>The process is waiting to read a message from a system queue that is empty. The PD is on the Read Queue List whose root is in the Queue Descriptor of the system queue involved.</td></tr> <tr> <td>07 Write Queue</td><td>The process is waiting to write a message to a system queue whose buffer is full. The PD is on the Write Queue List, whose root is in the Queue Descriptor of the system queue involved.</td></tr> </table>	00 RUN	The process is ready to run. The STAT field is always in this state when a process is examining its own Process Descriptor. The PD is on the Ready List. The currently running process is always at the head of Ready List.	01 POLL	The process is polling a device. The PD is on the Poll List.	02 DELAY	The process is delaying for a specified number of system ticks. The PD is on the Delay List.	06 Read Queue	The process is waiting to read a message from a system queue that is empty. The PD is on the Read Queue List whose root is in the Queue Descriptor of the system queue involved.	07 Write Queue	The process is waiting to write a message to a system queue whose buffer is full. The PD is on the Write Queue List, whose root is in the Queue Descriptor of the system queue involved.
00 RUN	The process is ready to run. The STAT field is always in this state when a process is examining its own Process Descriptor. The PD is on the Ready List. The currently running process is always at the head of Ready List.										
01 POLL	The process is polling a device. The PD is on the Poll List.										
02 DELAY	The process is delaying for a specified number of system ticks. The PD is on the Delay List.										
06 Read Queue	The process is waiting to read a message from a system queue that is empty. The PD is on the Read Queue List whose root is in the Queue Descriptor of the system queue involved.										
07 Write Queue	The process is waiting to write a message to a system queue whose buffer is full. The PD is on the Write Queue List, whose root is in the Queue Descriptor of the system queue involved.										

Table 6-17. (continued)

Field	Definition
08 FLAGWAIT	The process is waiting for a system flag to be set. The PD is in the flag table entry of the flag it is waiting for.
09 CIOWAIT	The process is waiting to attach to a character I/O device (console or list) while another process owns it. The PD is on CQUEUE list whose root is in the Character Control Block of the device in question.
PRIOR	<p>Current priority. Process scheduling is done based on this field. Typical user programs run at a priority of 200. 0 is the best priority, and 255 is the worst priority. The following is a list of priority assignments used by most Concurrent CP/M-86 systems. User processes priorities should be from 200-254.</p> <ul style="list-style-type: none"> <li>1 Initialization Process</li> <li>2 - 31 Interrupt Handlers</li> <li>32 - 63 System Processes</li> <li>64 - 189 Undefined</li> <li>191 - 197 Undefined</li> <li>198 Terminal Message Process</li> <li>199 Undefined</li> <li>200 Default Priority For Transients</li> <li>201 - 254 User Processes</li> <li>255 Idle Process</li> </ul>
FLAG	<p>Bit field of flags determining run-time characteristics of a process. Initialize as needed. All undocumented flags are used internally or are reserved for future use.</p> <p>001H SYS      System Process. Has privileged access to various features of Concurrent CP/M-86. This process can only be terminated if the termination code is 0FFH. This process can access restricted system queues. This flag is turned off if the calling process is not a system process.</p>



Table 6-17. (continued)

<i>Field</i>	<i>Definition</i>
002H KEEP	This process cannot be terminated. This flag is turned off if the calling process is not a system process.
004H KERNEL	This process resides within the operating system. This flag is turned off if the PD is not within the operating system.
010H TABLE	This PD is copied into the PD from the PD table. When this process terminates, the PD is recycled into the PD table.
NAME	Process Name. Eight bytes, all eight bits of each byte are used for matching process names.
UDA	Segment address of this processes User Data Area. Initialized to be the number of paragraphs from the beginning of the calling processes' Data Segment. The User Data Area contains process information that is not needed between processes. It also contains the System Stack of each process. Refer to the UDA description below.
DISK	Current default disk
USER	Current default user number
MEM	Root of linked list of Memory Segment Descriptors that are owned by this process. Initialized to zero, except for reentrant or shared code RSPs.
PARENT	Process that created this process. The P_CREATE system call sets this value at process creation. The parent field is set to zero if the parent terminates before the child.

Table 6-17. (continued)

<i>Field</i>	<i>Definition</i>
CNS	Current default console's number. Initialized to be the default console number.
LIST	Current default list device's. Initialized to be the default list device number.
RESERVED	Reserved for internal use. These fields must be initialized to zero (0).

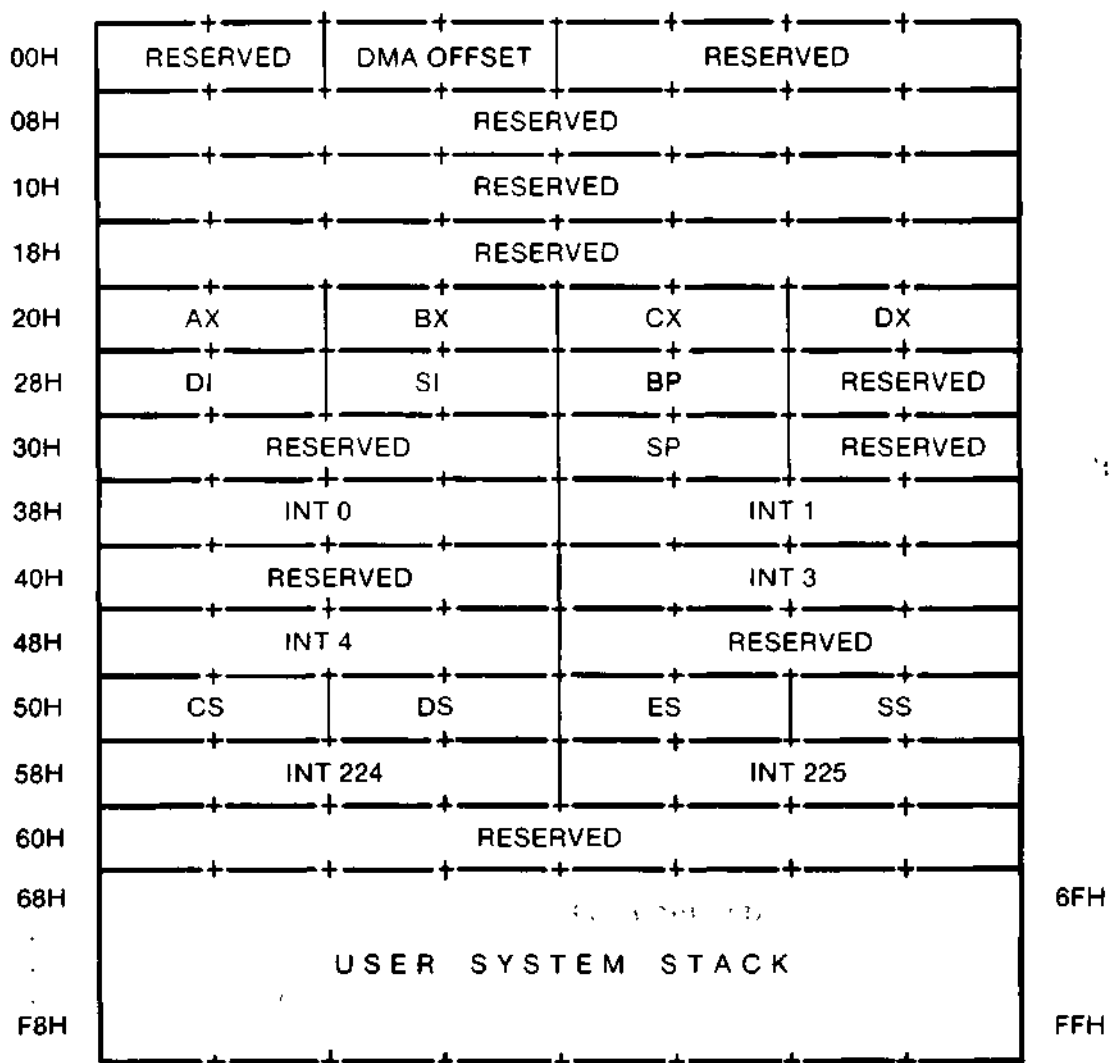


Figure 6-13. UDA - User Data Area

The length of the UDA is 256 bytes, and it must begin on a paragraph boundary.

Table 6-18. UDA Field Definition

Field	Definition												
DMA OFFS	The initial DMA offset for the new process. The segment address of the DMA is assumed to be the same as the initial Data Segment (refer to DS below).												
AX,BX,CX,DX, DI,SI,BP	The initial register values for the new process. These are typically set to zero.												
SP	<p>The initial stack pointer for the new process. The stack pointer is relative to the initial Stack Segment (refer to SS below). The initial stack of the new process must be initialized with the offset of the first instruction to be executed by the new process. The word that the stack pointer points to is the initial instruction pointer. Two words must follow the initial IP, which is filled in with the initial Code Segment (refer to CS below) and the initial flags. The initial flags are set to 0200H, which means that interrupts are on, and all other flags are off. Concurrent CP/M-86 starts a new process by executing an Interrupt Return instruction with the initial stack.</p> <p><b>Note:</b> this stack area is distinct from the User System Stack at the end of the UDA.</p> <div style="text-align: center;"><p>Low Memory</p><p>stack area</p><table><tr><td>SS</td><td>SP</td><td>IP</td><td></td></tr><tr><td></td><td></td><td>0</td><td>(CS)</td></tr><tr><td></td><td></td><td>0</td><td>(Flags)</td></tr></table><p>Stack Initialization Area</p></div>	SS	SP	IP				0	(CS)			0	(Flags)
SS	SP	IP											
		0	(CS)										
		0	(Flags)										

Table 6-18. (continued)

<i>Field</i>	<i>Definition</i>
INT 0, INT 1, INT 3, INT 4	The initial interrupt vectors for the first five interrupt types can be set by filling in these fields. The first word of each field is the Instruction Pointer (IP), and the second word is the Code Segment (CS) for a list of the interrupt routine that services these interrupts. Those fields that are zero are initialized to be the same as the calling processes interrupt vectors. These fields are typically initialized to be 0.
CS, DS, ES, SS	The initial segment addresses for the new process are taken from these fields. Those fields that are zero are initialized to be the same as the calling process's Data Segment.
INT 224, INT 225	Interrupts 224 and 225 are used to communicate with Concurrent CP/M-86 by typical programs. These interrupt vectors are initialized to be the same as the calling process if these values are zero. The ability to change these values allows a run-time system to intercept Concurrent CP/M-86 calls that its children make. The suggested protocol is to keep INT 225 pointing to the Concurrent CP/M-86 entry point and changing INT 224 to point to an internal routine. When a child process does an INT 224, the internal routine can filter calls to Concurrent CP/M-86 using INT 225 for the actual Concurrent CP/M-86 call.
RESERVED	These fields are used internally and must be initialized to zero.
USER SYSTEM STACK	This is the stack area used by the process when it is in the operating system. The SP variable in the UDA should not point to this area.

Refer to Table 6-5 for a list of error codes returned in CX.

P_DELAY	
Delay For Specified Number Of System Ticks	
Entry Parameters:	
Register	CL: 08DH (141)
	DX: Number of System Ticks

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 60Hz (16.67 milliseconds). In Europe, it is likely to be 50Hz (20 milliseconds). The exact length of the system tick can be obtained by reading the TICKSPERSEC 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.

P_DISPATCH	
Call Dispatcher	
Entry Parameters:	I
Register CL: 08EH (142)	

The P\_DISPATCH system call forces a reschedule of processes that are waiting to run. Normally, dispatches occur at every system tick interrupt (usually 60 times a second), and whenever a process releases a system resource. Dispatching also occurs whenever a process needs a system resource that is not currently available. A CPU-bound process runs for no more than one system tick before a dispatch is forced. The dispatch occurs at the next system tick.

The Concurrent CP/M-86 Dispatcher is priority driven, with round-robin scheduling of equivalent-priority processes. When a process calls the P\_DISPATCH system call, it is rescheduled, so that processes with higher or equivalent priorities are given the CPU before the calling process obtains it again. The calling process regains control of the CPU resource when it becomes the highest priority process again.

P_LOAD	
Load A CMD Type File Into Memory	
Entry Parameters:	
Register	CL: 03BH (59)
	DX: FCB Address - Offset
	DS: FCB Address - Segment
Returned Values:	
Register	AX: Base Page Segment
	0FFFFH on error
	BX: Same as AX
	CX: Error Code

The P\_LOAD system call loads a disk CMD type file into memory. Upon entry, register DX contains the offset, relative to DS, of a successfully opened FCB that specifies the CMD file to load. Upon return, register AX has the value 0FFFFH if the program load failed. Otherwise, AX contains the paragraph address of the Base Page belonging to the loaded program. The paragraph address and length of each group loaded from the CMD file, is found in the Base Page. See Sections 3.2 and 3.3.

Note that before calling P\_LOAD, the calling process must establish the DMA address of where the CMD file is to be loaded. This is accomplished with F\_DMA-SEG and F\_DMAOFF.

**Note:** open the CMD file in Read-Only mode and close it once the load is completed.

Refer to Table 6-5 for a list of error codes returned in CX.



P_PDADR	
Return The Address Of The Calling Process's Process Descriptor	
Entry Parameters:	
Register CL:	09CH (156)
Returned Values:	
Register AX:	PD Address - Offset
BX:	Same as AX
ES:	PD Address - Segment

The P\_PDADR system call obtains the address of the calling process's Process Descriptor. For a description of the format of the Process Descriptor, refer to the P\_CREATE system call.

P_PRIORITY	
Set The Priority Of The Calling Process	
Entry Parameters:	
Register	CL: 091H (145)
	DL: Priority

The P\_PRIORITY system call sets the priority of the calling process to the specified value. This system call is useful in situations where a process needs to have a high priority during an initialization phase, but afterwards can run at a lower priority.

The best or highest priority is 00H, while the worst or lowest priority is 0FFH. Transient processes are initialized to run at C8H (200 decimal) by the P\_CLI system call.

P_RPL	
Resident Procedure Library	
Entry Parameters:	
Register CL:	097H (151)
DX:	CPB Address - Offset
DS:	CPB Address - Segment
Returned Values:	
Register AX:	01H if RPL not found RPL return parameter
BX:	same as AX
CX:	Error Code
ES:	RPL return segment if addr

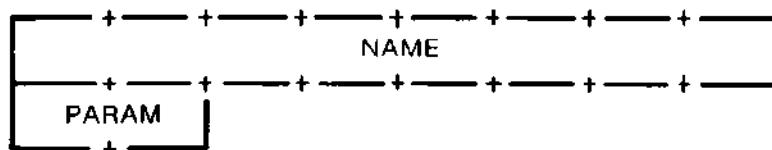


Figure 6-14. CPB - Call Parameter Block

Table 6-19. CPB Field Definitions

<i>Field</i>	<i>Definition</i>
NAME	Name of Resident Procedure, eight ASCII characters
PARAM	Parameter to send to the Resident Procedure

P\_RPL permits a process to call a system call in an optional Resident Procedure Library (RPL).

P\_RPL opens a system queue with the specified name. If the Q\_OPEN system call succeeds, P\_RPL checks the queue to verify that it is an RPL-type queue. If either the Q\_OPEN fails, or if it is not an RPL-type queue, P\_RPL returns to the calling process with an error code.

P\_RPL reads a message from the queue that contains the address of the specified system call. It then places the PARAM field of the CPB in register DX, and places the calling process's Data Segment address in register DS. P\_RPL performs a Far Call instruction to the address it obtains from the queue message. Upon return from the RPL, the system call copies the BX register to the AX register and then returns to the calling process.

**Note:** the P\_RPL system call does not write the address of the Resident Procedure back to the queue. The Resident Procedure itself must do this. If the Resident Procedure is to be reentrant, it must write the message into the queue upon entry. If it is to be serially reusable, the procedure must write the message just before returning.

Refer to Table 6-5 for a list of error codes returned in CX.

P_TERM	
Terminate Calling Process	
Entry Parameters:	
Register	AX: 0 on success 0FFFFH on failure
	BX: Same as AX
	CX: Error Code

The P\_TERM system call terminates the calling process. If the termination code is not 0FFH, the system call can only terminate a user process. If the termination code is 0FFH, the system call can terminate the calling process even though the process's System flag is on. P\_TERM cannot terminate a process with the KEEP flag on. If the termination is successful, the system call releases the mutual exclusion queues owned by the process. It also releases all memory segments owned by the process, and returns the Process Descriptor to the PD table.

A process can own one or more of the following resources: memory segments, consoles, printers, mutual exclusion messages, and system Lock List entries that record open files and locked records. When a process terminates and releases its resources, these resources become available to other processes on the system. For example, if a terminating process releases a system console, the console is usually given back to the console's TMP. This occurs when the TMP is the highest priority process waiting for the console.

P\_TERM does not return any results to the calling process. If the system call returns to the calling process, the P\_TERM call has failed for one of two reasons. Either the process has the KEEP flag on, or it has the SYSTEM flag on, and the termination code is not 0FFH.

P_TERMCPM	
Entry Parameters:	
Register CL:	00H (0)
Returned Values:	
Register AX:	0 on success 0FFFFH on failure
BX:	Same as AX
CX:	Error Code

The P\_TERMCPM system call terminates the calling process, releasing all system resources owned by the process.

P\_TERMCPM is implemented internally by calling P\_TERM with the termination code set to 00H.

Under CP/M-86, the P\_TERMCPM system call has a further argument that allows a process not to release its memory. This argument places a piece of code into memory that becomes an interface for later programs. Concurrent CP/M-86 does not include this option. Memory segments are not recovered by the system until all processes that own the memory segment have released it.

Refer to Table 6-5 for a list of returned error codes.

### 6.2.8 Queue System Calls

Queue system calls under Concurrent CP/M-86 use the Queue Parameter Block data structure to pass parameters to and from the operating system. Listing 6-3 shows the structure of the Queue Parameter Block and the equates for its fields.

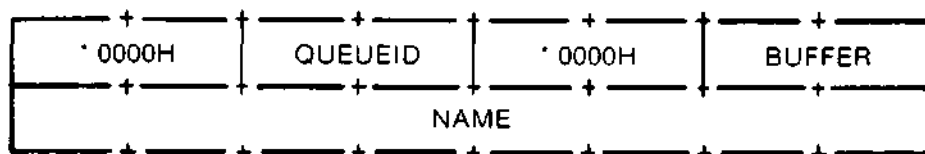


Figure 6-15. QPB - Queue Parameter Block

Table 6-20. QPB Field Definitions

<i>Field</i>	<i>Description</i>
QUEUEID	Queue number field; filled in by a Q_OPEN operation
* 0000H	Reserved for internal use; must be initialized to zero
BUFFER	Offset address of Queue Message Buffer
NAME	Name of Queue for Q_OPEN operation

```

;*****
;
;      QPB - Queue Parameter Block Definition
;
;      -
;
; 00  0000H  queueid  0000H  buffer
;
;      -
;
; 08                                name
;
;      -
;
;      queueid - Queue ID, address of QD
;      buffer - address to read/write into/from
;      name - name of queue (for open only)
;
;*****

```

```

qpb_0          equ      word ptr 0
qpb_queueid    equ      word ptr qpb_0 + word
qpb_buffer     equ      word ptr qpb_queueid + 4
qpb_name       equ      byte ptr qpb_buffer + word

qpb_len        equ      qpb_name + qnamsiz
qnamsiz        equ      8

```

Listing 6-3. Queue Parameter Block Definition



Q_CREAD	
Conditionally Read A Message From A System Queue	
<b>Entry Parameters:</b> Register CL: 08AH (138) DX: QPB Address - Offset DS: QPB Address - Segment QPB_queueid filled in by previous Q_OPEN QPB_buffer set to message buffer offset	
<b>Returned Values:</b> Register AX: 0 on success 0FFFFH on failure BX: Same as AX CX: Error Code message in buffer	

The Q\_CREAD system call is analogous to the Q\_READ system call, but it returns an error code if there are not enough messages to read, instead of waiting for another process to write to the queue.

Refer to Table 6-5 for a list of error codes returned in CX.

Q_CWRITE	
Conditionally Write A Message To A System Queue	
Entry Parameters:	
Register	CL: 08CH (140) DX: QPB Address - Offset DS: QPB Address - Segment QPB_queueid filled in by previous Q_OPEN QPB_buffer set to message buffer offset message in current DMA buffer
Returned Values:	
Register	AX: 0 on success 0FFFFH on failure BX: Same as AX CX: Error Code

The Q\_CWRITE system call is analogous to the Q\_WRITE system call, but it returns an error code if there is not enough system queue buffer space for the message to be written, instead of waiting for another process to read from the queue.

Refer to Table 6-5 for a list of error codes returned in CX.

Q_DELETE	
Delete A System Queue	
<b>Entry Parameters:</b>	
Register	CL: 088H (136)
	DX: QPB Address - Offset
	DS: QPB Address - Segment
	QPB_queueid filled in by a previous Q_OPEN call
<b>Returned Values:</b>	
Register	AX: 0 on success
	0FFFFH on failure
	BX: Same as AX
	CX: Error Code

The Q\_DELETE system call removes a system queue from the system. The system returns error codes if the queue cannot be deleted or if the queue has not been opened prior to the Q\_DELETE call.

Refer to Table 6-5 for a list of error codes returned in CX.

Q_MAKE	
Make A System Queue	
<b>Entry Parameters:</b> Register CL: 086H (134) DX: QD Address - Offset DS: QD Address - Segment QD filled in	
<b>Returned Values:</b> Register AX: 0 on success 0FFFFH on failure BX: Same as AX CX: Error Code	

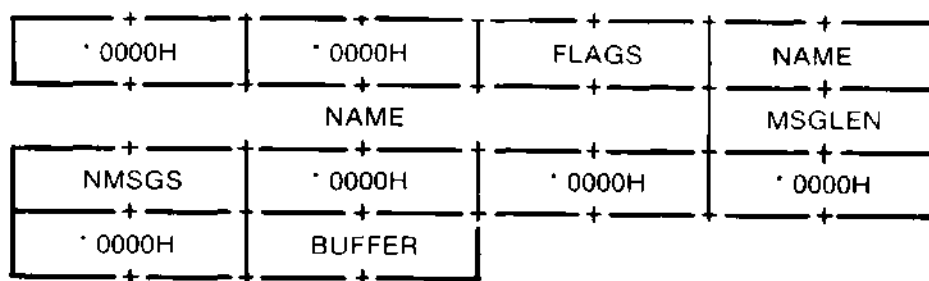


Figure 6-16. QD - Queue Descriptor

Table 6-21. Queue Descriptor Field Definitions

<i>Field</i>	<i>Definition</i>
FLAGS	<p>Queue Flags. The bits are defined as follows</p> <p>0001H - Mutual exclusion queue  0002H - Cannot be deleted  0004H - Restricted to system processes  0008H - RSP message queue  0010H - Used internally  0020H - RPL address queue  0040H - Used internally  0080H - Used internally</p> <p>Remaining flags reserved for future use</p>
NAME	8-byte queue name. All 8 bits of each character are matched on a Q_OPEN call.
MSGLEN	Number of bytes in each logical message
NMSGs	Maximum number of logical messages to be supported. If the number of messages written to the queue equals this maximum, no more messages are allowed until a message is read.
BUFFER	Address of the queue buffer. This buffer must be (NMSGs * MSGLEN) bytes long. The address is an offset relative to the DS register. This field is unused if the QD resides outside of the System Data Segment. Typically this field is 0 if the queue is being created by a transient program. RSPs that create queues must initialize this field to point to a buffer. The Data Segment of an RSP's queue is considered part of the System Data Segment unless it is beyond 64k of the beginning of the System Data Segment.
* 0000H	For internal use. Must be initialized to zero.

Every system queue under Concurrent CP/M-86 is associated with a Queue Descriptor that resides within the Concurrent CP/M-86 System Data Segment. In the Q\_MAKE system call, the calling process passes the address of a Queue Descriptor. If this Queue Descriptor is within the Concurrent CP/M-86 System Data Segment, the system uses it directly for the System Queue. If the Queue Descriptor is outside of the System Data Segment, the system obtains a Queue Descriptor from an internal Queue Descriptor table. If there are no unused Queue Descriptors in the internal table, the system call returns an error code.

Refer to Table 6-5 for a list of error codes returned in CX.

The buffer for a system queue must also reside within the System Data area. For non-00H length buffers, resident buffers are used directly. The system obtains a buffer from the Queue Buffer Area if the buffer does not reside within the System Data Segment. The size of the buffer is calculated from the NMSGs and MSGLEN fields. The system call returns an error code if there is not enough unused buffer area left to accommodate this new buffer.

All system queues must have unique names. The system call returns an error code if a system queue already exists by the given name.

Under Concurrent CP/M-86, all system queues must be explicitly opened (refer to the Q\_OPEN system call) before being used to read or write messages or to delete the queue.

Q_OPEN	
Open A System Queue	
Entry Parameters:	
Register	CL: 087H (135)
	DX: QPB Address - Offset
	DS: QPB Address - Segment
	QPB_name filled in
Returned Values:	
Register	AX: 0 on success
	0FFFFH on failure
	BX: Same as AX
	CX: Error Code
	QPB_queueid filled in

All system queues under Concurrent CP/M-86 must be explicitly opened before a read, write, or delete operation can be done. The Q\_OPEN system call examines each existing system queue and attempts to match the name in the QPB with the name of a system queue. All eight bytes of the name must match for a successful open. All bits of each byte are examined. If the open operation is successful, the Q\_OPEN system call modifies the Queue ID Field of the QPB. Once the the Queue is opened, subsequent reads, writes, or a delete are allowed.

Refer to Table 6-5 for a list of error codes returned in CX.

Q_READ	
Read A Message From A System Queue	
<b>Entry Parameters:</b>	
Register	CL: 089H (137)
	DX: QPB Address - Offset
	DS: QPB Address - Segment
	QPB_queueid filled in by previous Q_OPEN
	QPB_buffer set to message buffer offset
<b>Returned Values:</b>	
Register	AX: 0 on success
	0FFFFH on failure
	BX: Same as AX
	CX: Error Code message in buffer

The Q\_READ system call reads a message from a system queue that was previously opened by the calling process. The system call returns an error code if the queue was not previously opened or if the system queue has been deleted since the Q\_OPEN call. If there are not enough messages to read from the queue, the calling process waits until another process writes into the queue before returning.

Refer to Table 6-5 for a list of error codes returned in CX.



Q_WRITE	
Write A Message To A System Queue	
<b>Entry Parameters:</b>	
Register	CL: 08BH (139)
	DX: QPB Address - Offset
	DS: QPB Address - Segment
	QPB_queueid filled in by previous Q_OPEN
	QPB_buffer set to message buffer offset
	message in buffer
<b>Returned Values:</b>	
Register	AX: 0 on success
	0FFFFH on failure
	BX: Same as AX
	CX: Error Code

The Q\_WRITE system call writes a message to a system queue that was previously opened by the calling process. The system call returns an error code if the queue was not previously opened or if the system queue has been deleted since the Q\_OPEN call. If there is not enough buffer space in the queue, the calling process waits until another process reads from the queue before writing to the queue and returning.

Refer to Table 6-5 for a list of error codes returned in CX.

## 6.2.9 System Information System Calls

S_BDOSVER	
Return BDOS Version Number	
<p>Entry Parameters:</p> <p>Register CL: 0CH (12)</p> <p>Returned Values:</p> <p>Register AL: 31 (BDOS Version 3.1)</p> <p>AH: 14 (Concurrent CP/M-86)</p> <p>BX: Same as AX</p>	

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	
<p>0 = 8080</p> <p>1 = 8086</p>	
AH Low Nibble = OS Type	
0 = CP/M	2 = CP/M w/networking
1 = MP/M	3 = MP/M w/networking
4 = Concurrent CP/M	6 = Concurrent CP/M
5,7 to E = Reserved	w/networking

Figure 6-17. BDOS Version Number Format

S_BIOS	
Call BIOS Character Routine	
Entry Parameters:	
Register CL:	032H (50)
DX:	BIOS Desc. Addr. - Offset
DS:	BIOS Desc. Addr. - Segment
Returned Values:	
Register AX:	BIOS Return
BX:	Same as AX



Figure 6-18. BIOS Descriptor Format

The S\_BIOS system call is provided under Concurrent CP/M-86 for compatibility with programs generated under CP/M-86 that use this system call (Function 50). Under Concurrent CP/M-86, only routines that interface with character devices are supported. The arguments to character routines such as CONIN and LIST must be converted to those appropriate for the Concurrent CP/M-86 XIOS. Refer to the *Concurrent CP/M-86 System Guide* for further information about the XIOS.

**Note:** calls to the XIOS Console Status, Input, and Output system calls do not go to the XIOS if the referenced device is a virtual console.

S_OSVER	
Return The Version Of Current Concurrent CP/M-86 System	
Entry Parameters:	
Register CL:	0A3H (163)
Returned Values:	
AX:	Version Number (01410H)
BX:	Same as AX
CX:	Error Code

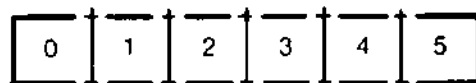
The S\_OSVER system call provides information that allows version-independent programming. The system call returns a two-byte value, with AH set to 014H for Concurrent CP/M-86, and AL set to the Concurrent CP/M-86 version level. The AH register contains a value set to the type of operating system. A value of 01420H indicates Concurrent CP/M-86 2.0.

Refer to Table 6-5 for a list of error codes returned in CX.

AL High Nibble = Concurrent CP/M-86 Version Number	
AL Low Nibble = Concurrent CP/M Revision Level	
AH High Nibble = CPU Type	
0 = 8080	:
1 = 8086	:
AH Low Nibble = OS Type	
0 = CP/M	2 = CP/M w/networking
1 = MP/M	3 = MP/M w/networking
4 = Concurrent CP/M	6 = Concurrent CP/M
5 7 to E = Reserved	w/networking

**Figure 6-19. Operating Systems Version Number Format**

S_SERIAL	
Return Current System's Serial Number	
Entry Parameters:	
Register	CL: 06BH (107)
	DX: SERIAL Address - Offset
	DS: SERIAL Address - Segment
Returned Values:	
	SERIAL filled in



**Figure 6-20. SERIAL Number Format**

S\_SERIAL returns the Concurrent CP/M-86 serial number to the addressed, six-byte SERIAL field as a six-byte ASCII numeral.

S_SYSDAT	
Return Address Of The System Data Segment	
Entry Parameters:	
Register CL:	09AH (154)
Returned Values:	
AX:	Sysdat Address - Offset
BX:	Same as AX
ES:	Sysdat Address - Segment

The S\_SYSDAT system call returns the address of the System Data Segment of the calling process. The System Data Segment contains all Process Descriptors, Queue Descriptors, the roots of system lists, and other internal data that Concurrent CP/M-86 uses.

Figure 6-21, illustrates the SYSDAT Table and its fields.

00H	SUP ENTRY				RESERVED			
08H	RESERVED							
10H	RESERVED							
18H	RESERVED							
20H	RESERVED							
28H	XIOS ENTRY				XIOS INIT			
30H	RESERVED							
38H	DISPATCHER				PDISP			
40H	CCPMSEG		RSPSEG		ENDSEG		RESER- VED	NVCNS
48H	NLCB	NCCB	N_ FLAGS	SYS_ DISK	MMP		RESER- VED	DAY FILE
50H	TEMP DISK	TICKS /SEC	LUL		CCB		FLAGS	
58H	MDUL		MFL		PUL		QUL	
60H	QMAU							
68H	RLR		DLR		DRL		PLR	
70H	RESERVED		THRDRT		QLR		MAL	
78H	VERSION		VERNUM		CCPMVERNUM		TOD_DAY	
80H	TOD _HR	TOD _MIN	TOD _SEC	NCON DEV	NLST DEV	NCIO DEV	LCB	
88H	OPEN_FILE		LOCK_ MAX	OPEN_ MAX				

Figure 6-21. SYSDAT Table



Table 6-22. SYSDAT Table Data Fields

<i>Field</i>	<i>Explanation</i>
SUP ENTRY	Double-word address of the Supervisor entry point for intermodule communication. All internal system calls go through this entry point.
XIOS ENTRY	Double-word address of the Extended I/O System entry point for intermodule communication. All XIOS function calls go through this entry point.
XIOS INIT	Double-word address of the Extended I/O System Initialization entry point. System hardware initialization takes place by a call through this entry point.
DISPATCHER	Double-word address of the Dispatcher entry point that handles interrupt returns. Executing a Far Jump to this address is equivalent to executing an Interrupt Return instruction. The Dispatcher routine causes a dispatch to occur and then executes an Interrupt Return. All registers are preserved and one level of stack is used. This location should be used as an exit point by all XIOS interrupt handlers that use the DEV_SETFLAG system call.
PDISP	Double-word address of the Dispatcher entry point that causes a dispatch to occur with all registers preserved. Once the dispatch is done, a RETF instruction is executed. Executing a JMPF PDISP is equivalent to executing a RETF instruction. This location should be used as an exit point whenever the XIOS releases a resource that might be wanted by a waiting process.
CCPMSEG	Starting paragraph of the operating system area. This is also the Code Segment of the Supervisor Module.

Table 6-22. (continued)

<i>Field</i>	<i>Explanation</i>
RSPSEG	Paragraph Address of the first RSP in a linked list of RSP Data Segments. The first word of the data segment points to the next RSP in the list. Once the system has been initialized, this field is zero.
ENDSEG	First paragraph beyond the end of the operating system area, including any buffers consisting of uninitialized RAM allocated to the operating system by GENCCPM. These include the Directory Hashing, Disk Data and XIOS ALLOC buffers. These buffer areas, however, are not part of the CCPM.SYS file.
NVCNS	Number of virtual consoles, copied from the XIOS Header by GENCCPM.
NLCB	Number of List Control Blocks, copied from the XIOS Header by GENCCPM.
NCCB	Number of Character Control Blocks, copied from the XIOS Header by GENCCPM.
NFLAGS	Number of system flags as specified during GENCCPM.
SYSDISK	Default system diskette. The CLI looks on this diskette if it cannot open the command file on the user's current default diskette. Set during GENCCPM.
MMP	Maximum memory allowed per process. Set during GENCCPM.
DAY FILE	Day File option. If this field is 0FFH, the operating system displays file logging information on system consoles at each command. Set during GENCCPM.

Table 6-22. (continued)

<i>Field</i>	<i>Explanation</i>
TEMP DISK	Default temporary diskette. Programs that create temporary files should use this diskette. Set during GENCCPM.
TICKS/SEC	The number of system ticks per second.
LUL	Link list root of unused Lock list items.
CCB	Address of the Character Control Block Table, copied from the XIOS Header by GENCCPM.
FLAGS	Address of the Flag Table.
MDUL	Link list root of unused Memory Descriptors.
MFL	Link list root of free memory partitions.
PUL	Link list root of unused Process Descriptors.
QUL	Link list root of unused Queue Descriptors.
QMAU	Queue Buffer Memory Allocation Unit.
RLR	Ready List Root. Linked list of PDs that are ready to run.
DLR	Delay List Root. Link list of PDs that are delaying for a specified number of system ticks.
DRL	Dispatcher Ready List. Temporary holding place for PDs that have just been made ready to run.
PLR	Poll List Root. Linked list of PDs that are polling on devices.

Table 6-22. (continued)

<i>Field</i>	<i>Explanation</i>
THRDRT	Thread List Root. Linked list of all current PDs on the system. The list is threaded though the THREAD field of the PD instead of the LINK field.
QLR	Queue List Root. Linked list of all System QDs.
MAL	Link list of active memory allocation units. A MAU is created from one or more memory partitions.
VERSION	Address, relative to CCPMSEG, of version string.
VERNUM	Concurrent CP/M-86 version number (system call 12, S_BDOSVER).
CCPMVERNUM	Concurrent CP/M-86 version number (system call 163, S_OSVER).
TOD_DAY	Time-of-Day. Number of days since 1 Jan, 1978.
TOD_HR	Time-of-Day. Hour of the day.
TOD_MIN	Time-of-Day. Minute of the hour.
TOD_SEC	Time-of-Day. Second of the minute.
NCONDEV	Number of XIOS consoles, copied from the XIOS Header by GENCCPM.
NLSTDEV	Number of XIOS list devices, copied from the XIOS Header by GENCCPM.
NCIODEV	Total number of character devices (NCONDEV + NLSTDEV).

Table 6-22. (continued)

<i>Field</i>	<i>Explanation</i>
LCB	Offset of the List Control Block Table, copied from the XIOS Header by GENCCPM.
OPEN_FILE	Open File Drive Vector. Designates drives that have open files on them. Each bit of the word value represents a disk drive; the least significant bit represents Drive A, and so on through the most significant bit, Drive P. Bits which are set indicate drives containing open files.
LOCK_MAX	Maximum number of locked records per process. Set during GENCCPM.
OPEN_MAX	Maximum number of open disk files per process. Set during GENCCPM.

T_GET	
Get System Time And Date	
Entry Parameters:	
Register	CL: 069H (105)
	DX: TOD Address - Offset
	DS: TOD Address - Segment
Returned Values:	
	AL: Seconds
	TOD filled in
	(Days, Hours and Minutes only)

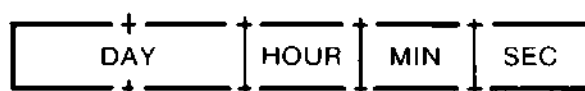


Figure 6-22. TOD - Time-of-Day Structure

Table 6-23. Time-of-Day Field Definitions

<i>Field</i>	<i>Definition</i>
DAY	The number of days since 1 January 1978. The day is stored as a 16-bit integer.
HOUR	The current hour of the current day. The hour is represented as a 24 hour clock in 2 binary coded decimal (BCD) digits.
MIN	The current minute of the current hour. The minute is stored as 2 BCD digits.
SEC	The current second of the current minute. The second is stored as 2 BCD digits.

The T\_GET system call obtains the system internal time and date. The calling process passes the address of a four-byte data structure that receives the time and date values. This system call is equivalent to the T\_SECONDS system call, except that it does not return the SECONDS field of the internal time.

T_SECONDS	
Get Current System Time And Day	
Entry Parameters:	
Register	CL: 09BH (155)
	DX: TOD Address - Offset
	DS: TOD Address - Segment
Returned Values:	
	TOD filled in (Days, Hours, Minutes, and Seconds)

The T\_SECONDS system call returns the current encoded time and date (including seconds) in the TOD structure passed by the calling process.



T_SET	
Set System Time And Date	
Entry Parameters:	
Register	CL: 068H (104)
	DX: TOD Address - Offset
	DS: TOD Address - Segment

The T\_SET system call sets the system internal time and date. The calling process passes the address of a 4-byte structure containing the time and date specification.

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 stored as two BCD digits.

Under Concurrent CP/M-86, this system call also sets the second field of the system time and date to 00H.

*End of Section 6*

1)

2)

3)

4)

# Appendix A

## Glossary

**Base Page:** Memory region between 0000H and 0100H relative to the beginning of the Data Segment used to hold system parameters. Base Page serves primarily as an interface region between user programs. Note that in the 8080 Model, the code and data are intermixed in the code segment.

**BCD:** Acronym for Binary Coded Decimal. Representation of decimal numbers using binary digits. See Table B-2 for binary representations of ASCII codes.

**BDOS:** Basic Disk Operating System (BDOS). The BDOS manages the Concurrent CP/M-86 file structure and executes most of the Concurrent CP/M-86 system calls.

**block:** Basic unit of disk space allocation under Concurrent CP/M-86. Each disk drive has a fixed block size (BLS) defined in its disk Parameter Block in the XIOS. The block size can be 1K, 2K, 4K, 8K, or 16K of consecutive bytes. Blocks are numbered relative to zero on a disk. Blocks are not shared between files.

**boolean:** Variable that can have only two values; usually interpreted as true/false or on/off.

**Checksum Vector (CSV):** Contiguous data area in the XIOS with one byte for each directory sector to be checked, that is, CKS bytes. A Checksum Vector is initialized and maintained for each logged-in drive. Each directory access by the system results in a checksum calculation that is compared with that in the Checksum Vector. If there is a discrepancy, the drive is set to Read-Only status. This prevents the user from inadvertently switching disks without logging in the new disk with a CTRL-C. If not logged in, the new disk is treated the same as the old one, and you can destroy data on it if you write to it.

**CIO:** Character I/O (CIO) Module. The CIO module handles all character I/O to and from consoles and list devices.

**CLI:** Command Line Interpreter. The P\_CLI system call interprets the command requested in a command line and performs the system calls needed to open a process, load the command file, and execute the code.

**CMD:** Filetype for Concurrent CP/M-86 command files. These are machine language object modules ready to be loaded and executed. Any file with this type can be executed by simply typing the filename after the drive prompt. For example, the program PIP.CMD can be executed by simply typing PIP.

**command:** Set of instructions that are executed when the command name is typed after the system prompt. These instructions can be built in the Concurrent CP/M-86 system or can reside on disk as a file of type CMD. Concurrent CP/M-86 commands consist of three parts: the command name, the command tail, and a carriage return.

**console:** Primary I/O device used by Concurrent CP/M-86. The console usually consists of a CRT screen for displaying output and a keyboard for input.

**control character:** Nonprinting ASCII character produced on the console by holding down the CTRL (CONTROL) key while striking the character key. CTRL-H means hold down CTRL and press H. Control characters are sometimes indicated using the up-arrow symbol (^), so CTRL-H can be represented as ^H. Certain control characters are treated as special commands by Concurrent CP/M-86.

**Default Buffer:** 128-byte buffer maintained at 0080H in the Base Page. When the CLI loads a CMD file, it initializes this buffer to the command tail, that is, any characters typed after the CMD file name. The first byte at 0080H contains the length of the command tail while the command tail itself begins at 0081H. A binary zero terminates the command tail value. The I command under DDT™ initializes this buffer in the same way as the CLI.

**Default FCB:** One of two FCBs maintained at 005CH and 006CH in the Base Page. The P\_CLI system call initializes the first default FCB from the first delimited field in the command tail and initializes the second default FCB from the next field in the command tail.

**delimiters:** ASCII characters used to separate constituent parts of a file specification. The P\_CLI system call recognizes certain delimiter characters as : . = ; < > \_ ' blank, and carriage return. Several Concurrent CP/M-86 commands also treat , [ ] ( ) , and \$ as delimiter characters. It is advisable to avoid the use of delimiter characters and lower-case characters in filenames.

**directory:** Portion of a disk containing entries for each file on the disk and locations of the blocks allocated to the files. Each file directory entry is in the form of a 32-byte FCB, although one file can have several entries, depending on its size. The maximum number of directory entries supported is specified in the drive's Disk Parameter Block.

**directory entry:** 32-byte entry associated with each disk file. A file can have more than one directory entry associated with it. There are four directory entries per directory sector. Directory entries can also be referred to as directory FCBs.

**disk, diskette:** Magnetic media used for mass storage of data in the computer system. The term disk can refer to a diskette, a removable cartridge disk, or a fixed hard disk.

**Disk Parameter Block (DPB):** Table residing in the XIOS that defines the characteristics of a drive in the disk subsystem used with Concurrent CP/M-86. The address of the DPB is in the Disk Parameter Header at DPbase + 0AH. Drives with the same characteristics can use the same DPB. However, each logical drive must have its own Disk Parameter Header and DPB. The address of the drive's Disk Parameter Header must be returned in registers HL when the BDOS calls the SELDSK entry point in the XIOS. DRV\_DPB returns the DPB address.

**Disk Parameter Header (DPH):** 16-byte area in the XIOS containing information about the disk drive and a scratchpad area for certain BDOS operations. See the *Concurrent CP/M-86 System Guide* for further details.

**extent (EX):** 16K consecutive bytes in a file. Extents are numbered from 0 to 31. One extent can contain 1, 2, 4, 8, or 16 blocks. EX is the extent number field of a FCB and is a one-byte field at FCB + 12, where FCB labels the first byte in the FCB. Depending on the Block Size (BLS) and the maximum data Block Number (DSM), a directory entry contains 1, 2, 4, 8, or 16 extents. The EX field is usually set to 0 by the user, but contains the current extent number during file I/O. The term "Extent Folding" describes directory entries containing more than one extent. In CP/M version 1.4, each FCB contained only one extent.

**FCB:** See File Control Block.

**file:** Collection of data containing from zero to 242,144 records. Each record contains 128 bytes and can contain either binary or ASCII data. Files consist of one or more 16K extents, with 128 records per extent.

**File Control Block (FCB):** Thirty-six consecutive bytes maintained and updated by system calls for file I/O. The FCB fields are described in Section 2.4.

**hex file format:** Absolute output of ASM86 for the Intel 8086. A HEX file contains a sequence of absolute records, which give a load address and byte values to be stored starting at the load address (refer to Section 4.3).

**I/O:** Acronym for Input/Output operations or routines handling the input and output of data in the computer system.

**logical drive:** Logically distinct region of a physical drive. A physical drive can be divided into one or more logical drives, and designated with specific drive references (such as d:a or d:f). Thus, at the user interface, it appears that there are several disks in the system.

**MEM:** Memory Module. The Memory Module handles all memory management calls by methods transparent to your applications program.

**parse:** Separate a command line into its syntactic parts.

**queue:** Data structure used by the file system to keep track system information, such as processes ready to run, locked files, and resources currently in use by processes. Processes also use queues to communicate with one another. The BDOS system calls create and maintain queues.

**Read-Only:** Condition in which a logical disk drive can be read but not written to. A drive can be set to Read-Only status by using the SET utility. This protects the user from switching disks without executing a disk reset. Files can also be set to Read-Only status with the SET utility or the F\_ATTRIB system call. Read-Only is often abbreviated as R/O.

**record:** Smallest unit of data in a disk file that can be read or written. A record consists of 128 consecutive bytes whose byte displacement in a file is the product of the Record Number times 128. A 128-byte record in a file occupies one 128-byte sector on the diskette. If the blocking and deblocking algorithm is used, several records can occupy each disk sector.

**reentrant code:** Code that can be used by one process while another is already executing it. Reentrant code must not be self-modifying; it must be pure code that does not contain data. The data for reentrant code can be kept in a separate data area or placed on the stack.

**RSP:** Reserved System Process. An RSP is a Concurrent CP/M-86 utility included within Concurrent CP/M-86 during the execution of GENCCPM.

**RTM:** Real Time Monitor. The RTM is the nucleus of Concurrent CP/M-86, managing queues and flags, polling devices, and dispatching and suspending processes. Application programs gain access to RTM functions through system calls.

**sector:** Unit of data read from and written to the disk by the XIOS. The sector size is dependent on the disk drive hardware and is usually a power of two, such as 256, 512, 1024, or 2048 bytes. These disk sectors are referred to as Host Sectors.

**source file:** ASCII text file usually created with a text editor that is an input file to a program, such as a compiler, assembler, or a text formatter.

**stack:** Reserved area of memory where the processor saves the return address when it receives a Call instruction. When the processor encounters a Return instruction, it restores the current address on the stack to the Instruction Pointer. Data such as the contents of the registers can also be saved on the stack on a first-in-last-out basis. The Push instruction places data on the stack and the Pop instruction removes it. 8086 stacks are 16 bits wide; instructions operating on the stack add and remove stack items one word at a time. An item is pushed onto the stack by decrementing the stack pointer (SP) by 2 and writing the item at the SP address. In other words, the stack grows downward in memory.

**SUP:** The Supervisor (SUP) manages communications between processes and the operating system kernel, and between other operating system modules. All system calls are intercepted by the SUP.

**track:** Concentric ring on the disk; the standard IBM.. single density disks have 77 tracks. Each track consists of a fixed number of numbered sectors. Tracks are numbered from 0 to one less than the number of tracks on the disk. Data on the disk media is accessed by combinations of track and sector numbers.

**TMP:** Terminal Message Processes. The TMPs are Resident System Processes that intercept command lines from the virtual consoles, check for errors, and pass on executable requests to the CLI. The TMP prints the prompt and some system error messages on your console. Each virtual console has an independent TMP heading defining the console's environment, including the default disk, user number, printer, and console.

**transient command file:** File of type .CMD stored on disk. Such files must be loaded into the system each time they are executed, and therefore execute more slowly than Resident System Processes (RSPs), which are an integral part of the operating system and execute rapidly. Transient commands are created with the GENCMD utility; RSPs are included in the operating system during execution of GENCCPM.

**user:** Logically distinct subdivision of the directory. Each directory can be divided into 16 user numbers.

**wildcard:** A ? or \* character. The BDOS directory search call matches ? with any single character and \* with multiple characters. Refer to the F\_SFIRST and F\_SNEXT system calls for further details.

**XIOS:** Extended I/O System. In Concurrent CP/M-86, the BDOS is the invariant file-handling system, which operates independent of the hardware implementation. The XIOS is the customizable I/O interface configured for your hardware system by the system manufacturer. The XIOS is similar to the BIOS in CP/M and CP/M-86, but it has been extended to implement virtual consoles and associated features.

*End of Appendix A*



# Appendix B

## ASCII and Hexadecimal Conversions

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

Table B-1. ASCII Symbols

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

Table B-2. ASCII Conversion Table

<i>Binary</i>	<i>Decimal</i>	<i>Hexadecimal</i>	<i>ASCII</i>
0000000	000	00	NUL
0000001	001	01	SOH (CTRL-A)
0000010	002	02	STX (CTRL-B)
0000011	003	03	ETX (CTRL-C)
0000100	004	04	EOT (CTRL-D)
0000101	005	05	ENQ (CTRL-E)
0000110	006	06	ACK (CTRL-F)
0000111	007	07	BEL (CTRL-G)
0001000	008	08	BS (CTRL-H)
0001001	009	09	HT (CTRL-I)
0001010	010	0A	LF (CTRL-J)
0001011	011	0B	VT (CTRL-K)
0001100	012	0C	FF (CTRL-L)
0001101	013	0D	CR (CTRL-M)
0001110	014	0E	SO (CTRL-N)
0001111	015	0F	SI (CTRL-O)
0010000	016	10	DLE (CTRL-P)
0010001	017	11	DC1 (CTRL-Q)
0010010	018	12	DC2 (CTRL-R)
0010011	019	13	DC3 (CTRL-S)
0010100	020	14	DC4 (CTRL-T)
0010101	021	15	NAK (CTRL-U)
0010110	022	16	SYN (CTRL-V)
0010111	023	17	ETB (CTRL-W)
0011000	024	18	CAN (CTRL-X)
0011001	025	19	EM (CTRL-Y)
0011010	026	1A	SUB (CTRL-Z)
0011011	027	1B	ESC (CTRL-[)
0011100	028	1C	FS (CTRL-\)
0011101	029	1D	GS (CTRL-])
0011110	030	1E	RS (CTRL-^)
0011111	031	1F	US (CTRL-_)
0100000	032	20	(SPACE)

Table B-2. (continued)

<i>Binary</i>	<i>Decimal</i>	<i>Hexadecimal</i>	<i>ASCII</i>
0100001	033	21	!
0100010	034	22	"
0100011	035	23	#
0100100	036	24	\$
0100101	037	25	%
0100110	038	26	&
0100111	039	27	'
0101000	040	28	(
0101001	041	29	)
0101010	042	2A	*
0101011	043	2B	+
0101100	044	2C	,
0101101	045	2D	-
0101110	046	2E	.
0101111	047	2F	/
0110000	048	30	0
0110001	049	31	1
0110010	050	32	2
0110011	051	33	3
0110100	052	34	4
0110101	053	35	5
0110110	054	36	6
0110111	055	37	7
0111000	056	38	8
0111001	057	39	9
0111010	058	3A	:
0111011	059	3B	;
0111100	060	3C	<
0111101	061	3D	=
0111110	062	3E	>
0111111	063	3F	?
1000000	064	40	@
1000001	065	41	A

Table B-2. (continued)

<i>Binary</i>	<i>Decimal</i>	<i>Hexadecimal</i>	<i>ASCII</i>
1000010	066	42	B
1000011	067	43	C
1000100	068	44	D
1000101	069	45	E
1000110	070	46	F
1000111	071	47	G
1001000	072	48	H
1001001	073	49	I
1001010	074	4A	J
1001011	075	4B	K
1001100	076	4C	L
1001101	077	4D	M
1001110	078	4E	N
1001111	079	4F	O
1010000	080	50	P
1010001	081	51	Q
1010010	082	52	R
1010011	083	53	S
1010100	084	54	T
1010101	085	55	U
1010110	086	56	V
1010111	087	57	W
1011000	088	58	X
1011001	089	59	Y
1011010	090	5A	Z
1011011	091	5B	[
1011100	092	5C	\
1011101	093	5D	]
1011110	094	5E	^
1011111	095	5F	<
1100000	096	60	'
1100001	097	61	a
1100010	098	62	b
1100011	099	63	c
1100100	100	64	d
1100101	101	65	e

Table B-2. (continued)

<i>Binary</i>	<i>Decimal</i>	<i>Hexadecimal</i>	<i>ASCII</i>
1100110	102	66	f
1100111	103	67	g
1101000	104	68	h
1101001	105	69	i
1101010	106	6A	j
1101011	107	6B	k
1101100	108	6C	l
1101101	109	6D	m
1101110	110	6E	n
1101111	111	6F	o
1110000	112	70	p
1110001	113	71	q
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	
1111101	125	7D	}
1111110	126	7E	-
1111111	127	7F	DEL

*End of Appendix B*

(

(

(

(

# Appendix C

## Error Codes

**Table C-1. Concurrent CP/M-86 Error Codes**

<i>Code#</i>	<i>Definition</i>
0	NO ERROR
1	FUNCTION NOT IMPLEMENTED
2	ILLEGAL FUNCTION NUMBER
3	CAN'T FIND MEMORY
4	ILLEGAL SYSTEM FLAG NUMBER
5	FLAG OVERRUN
6	FLAG UNDERRUN
7	NO UNUSED QUEUE DESCRIPTORS LEFT IN QD TABLE
8	NO UNUSED QUEUE BUFFER AREA LEFT
9	CAN'T FIND QUEUE
10	QUEUE IN USE
12	NO UNUSED PROCESS DESCRIPTORS LEFT IN PROCESS DESCRIPTOR TABLE
13	QUEUE ACCESS DENIED
14	EMPTY QUEUE
15	FULL QUEUE
18	NO UNUSED MEMORY DESCRIPTORS LEFT IN MEMORY DESCRIPTOR TABLE
19	ILLEGAL CONSOLE NUMBER
20	CAN'T FIND PROCESS DESCRIPTOR BY NAME
21	CONSOLE DOES NOT MATCH
22	NO CLI PROCESS
23	ILLEGAL DISK NUMBER
24	ILLEGAL FILE NAME
25	ILLEGAL FILE TYPE

Table C-1. (continued)

<i>Code#</i>	<i>Definition</i>
26	CHARACTER NOT READY
27	ILLEGAL MEMORY DESCRIPTOR
28	BAD LOAD
29	BAD READ
30	BAD OPEN
31	NULL COMMAND
32	NOT OWNER
33	NO CODE SEGMENT IN LOAD FILE
34	ACTIVE PROCESS DESCRIPTOR
35	CAN'T TERMINATE
36	CAN'T ATTACH
37	ILLEGAL LIST DEVICE NUMBER
38	ILLEGAL PASSWORD

*End of Appendix C*



# Appendix D

## ECHO.A86 Listing

```

;
;   ECHO - Resident System Process
;   Print Command tail to console
;

;
;   DEFINITIONS
;

ccpmint      equ      224      iccpm entry interrupt
c_writestr   equ      9        iprint string
c_detach     equ      147      idetach console
c_set        equ      148      iset default console
q_make       equ      134      icreate queue
q_open       equ      135      iopen queue
q_read       equ      137      iread queue
q_write      equ      139      iwrite queue
p_priority   equ      145      iset priority

pdlen        equ      48       ilength of Process
; Descriptor

p_cns        equ      byte ptr 020h   idefault cns
p_disk       equ      byte ptr 012h   idefault disk
p_user       equ      byte ptr 013h   idefault user
p_list       equ      byte ptr 024h   idefault list
ps_run       equ      0          ipd run status
pf_keep      equ      2          ipd nokill flag

rsp_top      equ      0          irsp offset
rsp_pd       equ      010h        ipd offset
rsp_uda      equ      040h        iuda offset
rsp_bottom   equ      140h        iend rsp header

rf_rsp       equ      08h         iqueue RSP flag

```

Listing D-1. ECHO.A86

```

;
; CODE SEGMENT
;
CSEG
org 0

ccpm:  int ccpmint
      ret

main:      ;create ECHO queue
      mov cl,q_make ! mov dx,offset qd
      call ccpm
      ;open ECHO queue
      mov cl,q_open ! mov dx,offset qpb
      call ccpm
      ;set priority to normal
      mov cl,p_priority ! mov dx,200
      call ccpm

      ;ES points to SYSDAT
      mov es,sdatseg

loop:    ;forever
      ;read cmdtail from queue
      mov cl,q_read ! mov dx,offset qpb
      call ccpm

      ;set default values from PD
      mov bx,PDadr
;      mov dl,es:p_disk[bx]      ;p_disk=0-15
;      inc dl ! mov disk,dl      ;make disk=1-16
;      mov dl,es:p_user[bx]
;      mov user,dl
;      mov dl,es:p_list[bx]
;      mov list,dl
      mov dl,es:p_cns[bx]
      mov console,dl

```

Listing D-1. (continued)

```

        iset default console
;      mov dl,console
        mov cl,C_SET | call ccpm

        iscan cmdtail and look for '$' or 0.
        iwhen found, replace w/ cr,lf,'$'

        lea bx,cmdtail | mov al,'$' | mov ah,0
        mov dx,bx | add dx,131
nextchar:
        cmp bx,dx | ja endcmd
        cmp [bx],al | je endcmd
        cmp [bx],ah | je endcmd
        inc bx | jmps nextchar
endcmd:
        mov byte ptr [bx],13
        mov byte ptr 1[bx],10
        mov byte ptr 2[bx],'$'

        iwrite command tail
        lea dx,cmdtail | mov cl,C_WRITESTR
        call ccpm

        idetach console
        mov dl,console
        mov cl,c_detach | call ccpm
        idone, set next command
        jmps loop

;
;      DATA SEGMENT
;

        DSEG
        org      rsp_top

sdatses      dw      0,0,0
              dw      0,0,0
              dw      0,0

```

Listing D-1. (continued)

```

018      rsp_pd

pd      dw      0,0          ; link,thread
        db      ps_run      ; status
        db      190         ; priority
        dw      pf_keep     ; flags
        db      'ECHO'      ; name
        dw      offset uda/10h ; uda seg
        db      0,0         ; disk,user
        db      0,0         ; load dsk,user
        dw      0           ; mem
        dw      0,0         ; duract,wait
        db      0,0
        dw      0
        db      0           ; console
        db      0,0,0
        db      0           ; list
        db      0,0,0
        dw      0,0,0,0

018      rsp_uda

uda     dw      0,offset dma,0,0      ;0
        dw      0,0,0,0
        dw      0,0,0,0              ;10h
        dw      0,0,0,0
        dw      0,0,0,0              ;20h
        dw      0,0,0,0
        dw      0,0,offset stack_tos,0 ;30h
        dw      0,0,0,0
        dw      0,0,0,0              ;40h
        dw      0,0,0,0
        dw      0,0,0,0              ;50h
        dw      0,0,0,0
        dw      0,0,0,0              ;60h

018      rsp_bottom

qbuf    rb      131              ;Queue buffer

```

Listing D-1. (continued)

```

qd          dw      0           ;link
            db      0,0         ;net,org
            dw      qf_rsp      ;flags
            db      'ECHO      ' ;iname
            dw      131         ;msglen
            dw      1           ;inmsgs
            dw      0,0         ;idq,inq
            dw      0,0         ;msgcnt,msgout
            dw      offset qbuf  ;ibuffer addr.

dma          rb      128

stack        dw      0ccccch,0ccccch,0ccccch
            dw      0ccccch,0ccccch,0ccccch
            dw      0ccccch,0ccccch,0ccccch
            dw      0ccccch,0ccccch,0ccccch
            dw      0ccccch,0ccccch,0ccccch

stack_tos    dw      offset main ; start offset
            dw      0           ; start seg
            dw      0           ; init flags

pdadr        rw      1           ; QPB Buffer
cmdtail      rb      128        ; starts here
            db      13,10,'$'

qpb          db      0,0         ;must be zero
            dw      0           ;queue ID
            dw      1           ;inmsgs
            dw      offset pdadr ;ibuffer addr.
            db      'ECHO      ' ;iname to open

console      db      0
idisk        db      0
iuser        db      0
ilist        db      0

            end

```

Listing D-1. (continued)

*End of Appendix D*

1

2

3

4

# Index

8080 and Small RSP Models  
    (Figure 5-1), 5-2  
8080 keyword, 4-8  
8080 Memory Model, 3-4, 3-6, 4-1,  
    4-2, 5-4  
96-byte initial stack, 3-1

## A

absolute address, 4-9  
ACB - Assign Control Block  
    (Figure 6-1), 6-22  
access stamp, 6-90  
AL0  
    AL1, 6-51  
Allocation Block Mask, 6-50  
Allocation Block Shift Factor, 6-50  
allocation vector, 6-47  
ambiguous reference, 6-106  
APB - Abort Parameter Block  
    (Figure 6-10), 6-46  
Archive attribute  
    T3', 2-16  
attribute bits, 2-15  
A\_Base, 3-4

## B

base extent, 6-101, 6-123  
Base Page Initialization, 3-4  
Base Page  
    8080 Model, 4-3  
    Compact Model, 4-6  
    initial data segment, 3-1  
    Small Model, 4-4

Basic Disk Operating System, 1-11,  
    2-1  
BDOS, 1-11  
BDOS Error mode, 6-79  
BDOS Multisector Count, 6-120  
BDOS physical errors, 2-47  
BDOS Version Number Format  
    (Figure 6-17), 6-182  
BIOS Descriptor Format  
    (Figure 6-18), 6-183  
bit map, 6-57  
BLM, 6-50  
blocking/deblocking, 2-42, 6-53  
BSH, 6-50  
burst mode, 2-37

## C

CCB, 1-12  
Character Control Block, 1-12, 6-156,  
    6-157  
checksum, 2-29, 6-89  
Checksum Vector Size, 6-52  
checksum verification, 2-29  
child process, 5-11  
CIO, 1-12  
CKS, 6-52  
CLI Command Line Buffer  
    (Figure 6-11), 6-150  
CLOCK process, 1-9  
Close Checksum error, 2-50  
closing files, 2-30  
CMD file, 1-14, 3-1, 5-1  
CMD File Header Format  
    (Figure 3-1), 3-2

- Code Group Descriptor, 5-2
- command file, 1-14
- Command Line Buffer, 6-151
- Command Line Interpreter, 2-6, 2-7
- Command RSP, 5-4
- Compact Memory Model, 3-3, 4-1, 4-6
- compatibility attribute definition, 2-34
- Concurrent CP/M-86 Base Page Values (Figure 3-3), 3-5
- Concurrent CP/M-86 Functional Modules (Figure 1-2), 1-3
- Concurrent CP/M-86 Small Memory 3-4, 4-1, 4-4
- Concurrent CP/M-86 Virtual/Physical Environments (Figure 1-1), 1-1
- conditional read, 1-8
- conditional write, 1-8
- Console Buffer Format (Figure 6-2), 6-34
- contiguous memory segment, 6-138
- control characters, 6-33
- converting 8080 programs to Concurrent CP/M-86, 4-9
- CPB - Call Parameter Block (Figure 6-14), 6-167
- CR field, 3-7
- CTRL-Z at EOF, 2-10
- current record field, 6-88
- current record position, 3-7
- current user number, 6-119
- CX error codes, 1-15
- C\_ASSIGN system call, 6-22
- C\_ATTACH system call, 6-24
- C\_CATTACH system call, 6-25
- C\_DELIMIT system call, 6-26
- C\_DETACH system call, 6-27
- C\_GET system call, 6-28

- C\_MODE system call, 6-29
- C\_RAWIO system call, 6-31
- C\_READ system call, 6-33
- C\_READSTR system call, 6-34
- C\_SET system call, 6-37
- C\_STAT system call, 6-38
- C\_WRITE system call, 6-39
- C\_WRITEBLK system call, 6-40
- C\_WRITESTR system call, 6-41

## D

- data area, 2-1
- data block size, 2-9
- Data Group Descriptor, 5-2, 5-12
- default DMA base, 6-78
- default DMA buffer, 3-6
- default drive, 3-5
- Default Error mode, 2-48
- Delay List, 6-155
- Delete mode, 2-23
- delimiters, 2-6, 6-93
- detach character, 6-33
- DEV\_POLL system call, 1-13, 6-42
- DEV\_SETFLAG system call, 6-43
- DEV\_WAITFLAG system call, 1-9, 6-44
- DIR utility, 2-1
- Direct Memory Address, 6-77
- Directory Allocation Vectors, 6-51
- directory area, 2-1
- directory code, 2-51, 2-53
- directory label, 2-3, 2-19, 2-20, 2-56, 2-61
- directory label data byte, 6-56, 6-61
- Directory Label Format (Figure 2-2), 2-19
- Directory Maximum, 6-51



- Directory Record with SFCB
  - (Figure 2-4), 2-25
- disk directory area, 2-9
- Disk Free Space Field Format
  - (Figure 6-5), 6-66
- Disk I/O error, 2-47
- Disk Parameter Block, 2-44, 6-50
- Disk Reset, 2-45, 6-52
- Disk Storage Maximum, 6-51
- Disk System Reset (Figure 2-6), 2-45
- Dispatcher, 1-6
- DMA address, 3-1
- DMA base, 3-1
- DMA Buffer, 5-7, 5-10, 6-77
- DMA offset, 3-1, 6-76, 6-160
- DMA
  - default address, 6-48
- DPB, 2-44, 6-49
- DPB - Disk Parameter Block
  - (Figure 6-4), 6-50
- drive capacity, 2-9
- drive specifier, 2-5
- drives, 2-9
  - drive-related system calls, 2-3
- DRM, 6-51
- DRV\_ACCESS system call, 2-46, 6-46
- DRV\_ALLOCVEC system call, 6-47
- DRV\_ALLRESET system call, 2-43, 3-1, 6-48
- DRV\_DPB system call, 6-49
- DRV\_FLUSH system call, 6-53
- DRV\_FREE, 2-32
- DRV\_FREE system call, 2-46, 6-54
- DRV\_GET system call, 6-55
- DRV\_GETLABEL system call, 6-56
- DRV\_LOGINVEC system call, 6-57
- DRV\_RESET system call, 2-44, 6-58
- DRV\_ROVEC system call, 6-59
- DRV\_SET system call, 6-60

- DRV\_SETLABEL system call, 2-21, 6-61
- DRV\_SETRO system call, 2-47, 2-49, 6-64
- DRV\_SPACE system call, 6-65
- DSM, 6-51

## E

- ECHO, 5-3, 5-8, 5-12
- EOF (CTRL-Z), 2-10
- error codes, 1-15, 2-50, 2-51
- error flag, 2-51, 2-54
- error handling, 2-47
- Error Mode, 2-3
- EXM, 6-51
- extended errors, 2-47, 2-48, 2-50
- extent, 6-98, 6-120
- Extent Mask, 6-51

## F

- Far Return, 3-1, 4-3
- FCB - File Control Block (Figure 2-1), 2-11
  - checksum, 2-29
  - format, 2-18
  - length, 2-11
  - verification, 2-46
    - File Name1, 3-6
    - File Name2, 3-7
- file access, 2-38
- File Already Exists error, 2-50
- file attributes, 6-67
  - set/reset, 2-15
- File Control Block, 2-10
- File Currently Open error, 2-50

File ID, 2-13, 2-28, 2-39, 6-80, 6-89, 6-116  
 file locking, 1-11  
 file naming conventions, 2-5  
 File Opened in Read/Only Mode error, 2-50  
 file references, 2-1  
 file security, 2-29  
 file size, 2-9  
 file specification, 2-5  
 file system, 2-3, 2-5, 2-9, 2-30, 2-41, 2-42  
 file-access system calls, 2-2  
 filename field, 2-1, 2-5  
 filetype field, 2-1, 2-5  
 Flag 1  
     the system tick flag, 1-10  
 flush buffers, 2-43  
 Function 0, 6-170  
 Function 1, 6-33  
 Function 2, 6-39  
 Function 5, 6-134  
 Function 6, 6-31  
 Function 9, 6-41  
 Function 10, 6-34  
 Function 11, 6-38  
 Function 12, 6-182  
 Function 13, 6-48  
 Function 14, 6-60  
 Function 15, 6-88  
 Function 16, 6-70  
 Function 17, 6-106  
 Function 18, 6-110  
 Function 19, 6-73  
 Function 20, 6-98  
 Function 21, 6-120  
 Function 22, 6-84  
 Function 23, 6-104  
 Function 24, 6-57  
 Function 25, 6-55  
 Function 26, 6-77  
 Function 27, 6-47  
 Function 28, 6-64  
 Function 29, 6-59  
 Function 30, 6-67  
 Function 31, 6-49  
 Function 32, 6-119  
 Function 33, 6-101  
 Function 34, 6-123  
 Function 35, 6-108  
 Function 36, 6-97  
 Function 37, 6-58  
 Function 38, 6-46  
 Function 39, 6-54  
 Function 40, 6-128  
 Function 42, 6-80  
 Function 43, 6-116  
 Function 44, 6-87  
 Function 45, 6-79  
 Function 46, 6-65  
 Function 47, 6-149  
 Function 48, 6-53  
 Function 50, 6-183  
 Function 51, 6-78  
 Function 52, 6-76  
 Function 53, 6-145  
 Function 54, 6-140  
 Function 55, 6-142  
 Function 56, 6-143  
 Function 57, 6-144  
 Function 58, 6-141  
 Function 59, 6-164  
 Function 99, 6-113  
 Function 100, 6-61  
 Function 101, 6-56  
 Function 102, 6-111  
 Function 103, 6-126  
 Function 104, 6-197  
 Function 105, 6-194  
 Function 106, 6-96  
 Function 107, 6-186  
 Function 109, 6-29

Function 110, 6-26  
 Function 111, 6-40  
 Function 112, 6-135  
 Function 128, 6-138  
 Function 129, 6-138  
 Function 130, 6-139  
 Function 131, 6-44  
 Function 132, 6-43  
 Function 133, 6-42  
 Function 134, 6-176  
 Function 135, 6-179  
 Function 136, 6-175  
 Function 137, 6-180  
 Function 138, 6-173  
 Function 139, 6-181  
 Function 140, 6-174  
 Function 141, 6-162  
 Function 142, 6-163  
 Function 143, 6-169  
 Function 144, 6-153  
 Function 145, 6-166  
 Function 146, 6-24  
 Function 147, 6-27  
 Function 148, 6-37  
 Function 149, 6-22  
 Function 150, 6-150  
 Function 151, 6-167  
 Function 152, 6-91  
 Function 153, 6-28  
 Function 154, 6-187  
 Function 155, 6-196  
 Function 156, 6-165  
 Function 157, 6-146  
 Function 158, 6-129  
 Function 159, 6-131  
 Function 160, 6-133  
 Function 161, 6-130  
 Function 162, 6-25  
 Function 163, 6-184  
 Function 164, 6-132  
 F\_ATTRIB system call, 2-15, 6-67

F\_CLOSE system call, 6-70  
 F\_DELETE system call, 6-73  
 F\_DMAGET system call, 6-76  
 F\_DMAOFF system call, 5-7, 6-77  
 F\_DMASEG system call, 5-7, 6-78  
 F\_ERRMODE system call, 6-79  
 F\_FLUSH system call, 2-43  
 F\_LOCK system call, 6-80  
 F\_MAKE system call, 2-11, 2-15, 2-22,  
     2-29, 6-84  
 F\_MULTISEC system call, 2-38, 6-87  
 F\_OPEN system call, 2-11, 2-15, 2-27,  
     2-28, 2-29, 6-88  
 F\_PARSE system call, 2-7, 3-1, 6-91,  
     6-151  
 F\_PASSWD system call, 6-96  
 F\_RANDREC system call, 6-97  
 F\_READ system call, 6-98  
 F\_READRAND system call, 6-101  
 F\_RENAME system call, 6-104  
 F\_SFIRST system call, 6-106  
 F\_SIZE system call, 6-108  
 F\_SNEXT system call, 6-110  
 F\_TIMEDATE system call, 6-111  
 F\_TRUNCATE system call, 6-113  
 F\_UNLOCK system call, 6-116  
 F\_USERNUM system call, 6-119  
 F\_WRITE system call, 6-120  
 F\_WRITERAND system call, 6-123  
 F\_WRITEXFCB system call, 2-23,  
     6-126  
 F\_WRITEZF system call, 6-128

## G

G-Form, 3-2  
 GENCCPM, 5-1, 5-10  
 GENCMD, 4-8, 4-11, 5-2  
 Group Descriptor, 3-2

Group Descriptor Format (Figure 3-2),  
3-3

G\_Length, 3-4

G\_Max, 3-4

G\_Min, 3-4

## H

Hard Disk, 6-52

header record, 3-2

CMD file, 4-1, 4-9

## I

Illegal ? in FCB error, 2-51

independent group, 3-6

initial stack, 4-7

8080 model, 4-2

Instruction Pointer, 4-3, 6-161

INT 224, 1-15, 6-161

INT 225, 6-161

Intel Hexadecimal File Format, 4-11

Intel utilities, 4-9

interface attribute F5', 6-70, 6-73, 6-88

interface attribute F6', 6-70, 6-88

interface attribute f7', 6-89

interface attribute f8', 6-89

interface attributes, 2-17, 2-18, 2-29

intermodule communications, 1-5

Invalid Drive error, 2-49

IRET instruction, 5-9, 5-10

IRET structure, 5-9, 5-11

## K

KEEP flag, 6-169

## L

Lock List, 2-29, 2-30, 2-31, 2-41, 2-46,  
6-81

Locked mode, 2-27

log-in operation, 2-44

logical drive, 2-1, 2-8

login vector, 6-57

L\_ATTACH system call, 6-129

L\_CATTACH system call, 6-130

L\_DETACH system call, 6-131

L\_GET system call, 6-132

L\_SET system call, 6-133

L\_WRITE system call, 6-134

L\_WRITEBLK system call, 6-135

## M

M80 byte, 3-6

maximum memory size, 4-10

MCB - Memory Control Block  
(Figure 6-7), 6-135

MC\_ABSALLOC system call, 6-140

MC\_ABSMAX system call, 6-141

MC\_ALLFREE system call, 6-142

MC\_ALLOC system call, 6-143

MC\_FREE system call, 6-144

MC\_MAX system call, 6-145

media change, 2-46

Media

Nonremovable, 6-52

MEM, 1-10

Memory Control Block, 6-135

memory models, 4-1

Memory Module, 1-10

memory protection, 6-154

memory, 3-7

absolute, 6-140

initialization, 3-1

largest available region, 6-145

**MFPB - M\_FREE Parameter Block**  
 (Figure 6-9), 6-139  
 minimum memory value, 4-9  
**MPB - Memory Parameter Block**  
**Multisector Count**, 2-3, 2-38, 6-77,  
 6-80, 6-87, 6-98, 6-102, 6-103,  
 6-124, 6-125  
 I/O, 2-37  
 mutual exclusion queues, 1-8  
**MX queue**, 1-8  
**MXdisk**, 1-9  
**M\_ALLOC** system call, 6-138  
**M\_FREE** system call, 6-139

## N

networking interfaces, 1-5  
**No Room In System Lock List error**,  
 2-51  
 not-8080 model, 3-6  
**Nonremovable Media Drives**, 6-52

## O

**OFF**, 6-52  
 one second flag  
 Flag 2, 1-9  
**Open File Limit Exceeded error**, 2-51  
 open mode, 2-27  
**Operating System Version Number**  
 Format (Figure 6-19), 6-185

## P

**P1 Len**, 3-6  
**P2 Len**, 3-6  
 parent/child relationship, 3-7  
 parse filename, 2-7

**Parse Filename Control Block**, 6-91  
 password, 2-1, 2-23, 2-24  
 password 1 address, 3-6  
 password 2 address, 3-6  
**Password error**, 2-50  
 field, 2-5  
 length, 3-6  
 protection, 2-23  
**PD - Process Descriptor** (Figure 6-12),  
 6-154  
 permanent drive, 2-44, 2-47  
**PFCB - Parse Filename Control Block**  
 (Figure 6-6), 6-91  
 physical error, 2-47  
 physical error codes, 2-55  
**Physical Record Shift Factor**, 6-52  
**Poll List**, 6-155  
 printer echo, 6-33  
 priority levels, 1-7  
 priority, 6-156, 6-166  
 transient process, 5-4  
 process, 1-2, 2-30, 2-37  
**Process Descriptor**, 1-6, 1-9, 6-154  
 initialization, 3-1  
 process priority, 6-162  
**PSH**, 6-52  
**P\_ABORT** system call, 6-146  
**P\_CHAIN** system call, 2-18, 6-149  
**P\_CLI** system call, 2-18, 3-1, 6-33,  
 6-150  
**P\_CREATE** system call, 3-1, 5-1,  
 6-153  
**P\_DELAY** system call, 1-10, 6-162  
**P\_DISPATCH** system call, 6-163  
**P\_LOAD** system call, 3-4, 4-2, 6-164  
**P\_PDADR** system call, 5-6, 6-165  
**P\_PRIORITY** system call, 6-166  
**P\_RPL** system call, 6-167  
**P\_TERM** system call, 3-1, 6-33,  
 6-169  
**P\_TERMCPM** system call, 6-170

## Q

QD - Queue Descriptor (Figure 6-16), 6-176  
QPB - Queue Parameter Block (Figure 6-15), 6-171  
qualified reset, 2-44  
Queue Buffer, 1-8  
Queue Descriptor, 1-9, 6-176  
    RSP Command Queue, 5-10  
Queue Parameter Block, 5-11  
Q\_CREAD system call, 5-5, 6-173  
Q\_CWRITE system call, 6-174  
Q\_DELETE system call, 5-10, 6-175  
Q\_MAKE system call, 5-10, 6-176  
Q\_OPEN system call, 6-179  
Q\_READ system call, 1-8, 5-5, 6-180  
Q\_WRITE system call, 6-181

## R

R/O drive reset, 2-45  
R/O Vector, 6-59  
Random Record Number, 2-10, 3-7, 6-80, 6-97, 6-101, 6-108, 6-116, 6-123  
Read mode, 2-23  
Read Queue List, 6-155  
Read-Only attribute T1', 2-16, 6-89  
Read-Only mode, 2-28, 2-38  
Read/Only Disk error, 2-49  
Read/Only File error, 2-49  
Ready List, 1-6, 6-155  
ready process, 1-6  
Real-Time Monitor, 1-5  
record locking, 2-30, 2-39  
register AL, 2-53  
register contents preserved, 1-15  
register initialization, 5-10  
registers used by system calls, 1-15

removable drive, 2-44, 2-47  
reset permanent drive, 2-44  
reset removable drive, 2-44  
reset state, 6-48, 6-58  
Resident Procedure Library, 6-167  
Resident System Processes, 5-1, 6-151  
Return and Display Error mode, 2-48, 6-79  
return codes, 2-51  
Return Error mode, 2-48, 6-79  
RSP Command Queue, 5-4  
RSP Command Queue Message (Figure 5-3), 5-5  
RSP copies  
    8080 Models, 5-4  
    Small Model, 5-4  
RSP Data Segment (Figure 5-4), 5-7  
RSP Header, 5-3, 5-7  
RSP Header Format (Figure 5-2), 5-3  
RSP Memory Models, 5-1  
RSP Process Descriptor, 5-5, 5-6, 5-7, 5-8, 5-9  
RSP Stack, 5-9  
RSP UDA, 5-7  
RSP User Data Area, 5-9  
RSP  
    8080 Memory Model, 5-2  
    CMD Header Record, 5-2  
    ECHO, 5-1  
    multiple copies, 5-3  
    Process Descriptor, 5-1  
    shared code, 5-4  
    Small Memory Model, 5-2  
    TMP, 5-3  
    UDA, 5-1  
RTM, 1-5  
RUN state, 6-43, 6-44, 6-175

## S

- Sectors Per Track, 6-50
- segment group memory requirements, 4-8
- segment register change, 4-6
- segment register initialization, 4-2
- sequential I/O processing, 2-38
- SERIAL Number Format (Figure 6-20), 6-186
- SFCB Subfields (Figure 2-5), 2-26
- Shared access to files, 1-11
- Small Memory Model, 3-4, 4-2, 4-4
- sparse files, 2-10
- SPT, 6-50
- Stack Pointer, 6-160
- stamp, 6-86
- start scroll character, 6-30
- stop scroll character, 6-30
- SUP, 1-5
- Supervisor, 1-5
- suspended process, 1-6
- SYSDAT Table (Figure 6-21), 6-188
- system attribute T2', 2-16, 6-189
- system call calling conventions, 1-15
- system call register initialization, 1-15
- System Data Area, 5-6, 5-8, 5-9
- System Data Segment, 6-153, 6-178
- SYSTEM flag, 6-169
- system generation, 5-1
- system Lock List, 6-46, 6-54
- system performance, 1-7
- system process
  - TICK, 1-9
- system processes, 1-2

- System Queue, 1-8, 6-178
- System Tick, 6-162, 6-163
- system timing, 1-9
- S\_BDOSVER system call, 6-182
- S\_BIOS system call, 6-183
- S\_OSVER system call, 6-184
- S\_SERIAL system call, 6-186
- S\_SYSDAT system call, 6-187

## T

- tab character, 6-33
- termination character, 6-33
- Termination Code, 6-147, 6-169
- time-of-day, 1-9, 6-194
- time stamping, 2-3
- TOD - Time-of-Day Structure (Figure 6-22), 6-194
- Track Offset, 6-52
- Transient Process Area, 6-153
- transient processes, 1-2, 1-5
- transient programs, 1-14
- T\_GET system call, 6-194
- T\_SECONDS system call, 6-196
- T\_SET system call, 6-197

## U

UDA - User Data Area (Figure 6-13),  
6-159

UDA, 1-6

initialization, 3-1

unconditional read, 1-8

Unlocked mode, 2-28, 2-38

User 0, 2-19

User Data Area, 6-153, 6-159

user directories, 2-18

user number, 2-1, 2-18

## V

virtual file size, 6-108

## W

Write mode, 2-23

Write Queue List, 6-155

## X

XFCB, 2-21

XFCB - Extended File Control Block  
(Figure 2-3), 2-21

XFCB password mode, 6-111, 6-126

XIOS, 1-13