This revised paperback edition of the successful CP/M Plus Handbook now contains a new section on GSX Graphics.

'It is a substantial book by any standard, and an absolute must for the programmer-user...it is comprehensive and authoritative, two things that cannot be said about many computer books. The CP/M Plus Handbook is packed with solid microcomputing muscle.'

Practical Computing

'For a beginner to CP/M Plus, the handbook makes interesting reading, providing a valuable insight into the working of the computer. Although the second part is recommended for those with previous knowledge it, too, contains plenty to interest beginners and experts alike.'

Putting Your Amstrad to Work

ISBN 0-434-90321-3
9 780434 903214

CP/M PLUS HANDBOOK Digital Research Inc. Amstrad Consumer Electronics plc.

CP/M PLUS HANDBOOK

OPERATOR'S AND PROGRAMMER'S GUIDE FOR THE AMSTRAD CPC6128 AND PCW8256

```
A) DIS
Insert first
Insert second di
Strike any key
Comparing 9 se
Diskettes com
          more
```

Authors:
Digital Research Inc.
Amstrad Consumer Electronics plc.

NOW WITH GSX SUPPLEMENT

HEINEMANN NEWTECH

REVISED PAPERBACK EDITION

The Digital Research CP/M Plus Manual

The Digital Research CP/M Plus Manual

for Amstrad PCW8256
and Amstrad CPC6128

Heinemann: London

William Heinemann Ltd
10 Upper Grosvenor Street, London W1X 9PA

LONDON MELBOURNE
JOHANNESBURG AUCKLAND

First edition January 1983 (Digital Research Inc.)
Second edition March 1983 (Digital Research Inc.)
First published in this edition by Heinemann 1986.
Second edition published in paperback 1987

British Library Cataloguing in Publication Data

The Digital Research CP/M Plus Manual
1. CP/M Plus (Computer operating system)
I. Digital Research
005.4'46 QA76.76.063

ISBN 0 434 90321 3

Typeset by Express Typesetters Ltd, Aldershot
Printed in Great Britain by Redwood Burn Limited, Trowbridge, Wiltshire

CONTENTS

Disclaimer

Notice to User

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

Trademarks

PREFACE

This guide provides you with a full description of the CP/M Plus operating system for your Amstrad PCW8256 or CPC6128. CP/M Plus is also known as CP/M 3.

The PCW8512 is a variant of the PCW8256 (with 512K memory and second disk drive): for the purposes of this manual the 8512 is treated as for the 8256.

This guide is divided into two main parts, as follows;

Part 1	Using CP/M Plus. This is a full specification of the commands and facilities of CP/M Plus and should be used in conjunction with the User Guide supplied with your computer.
Part 2	Programming CP/M Plus. This describes the programming environment of CP/M Plus. Note that this information is designed for experienced programmers who are familiar with the writing of application software.

What CP/M Plus Does For You

CP/M Plus manages and supervises your computer's resources, including memory and disk storage, the console (screen and keyboard), printer, and communications devices. It also manages information stored magnetically on disks by grouping this information into files of programs or data. CP/M Plus can copy files from a disk to your computer's memory, or to a peripheral device such as a printer. To do this, CP/M Plus places various programs in memory and executes them in response to commands you enter at your console.

Once in memory, a program executes through a set of steps that instruct your computer to perform a certain task. You can use CP/M Plus to create your own programs, or you can choose from the wide variety of CP/M Plus application programs that entertain you, educate you, and help you solve commercial and scientific problems.

CP/M Plus on CPC6128 and PCW8256

This guide is specifically designed for users of the Amstrad PCW8256 and CPC6128 machines. Differences between the machines are highlighted in the text. Note that the CPC6128 can support two disk drives while the PCW8256 can support two drives plus a memory drive (M:). Details of operating the equipment are given in your User Guide.

Part 1

USING CP/M PLUS

Section 1
Introduction to CP/M Plus

This section tells you how to start CP/M Plus, how to enter and edit the command line, and how to make back-up copies of your CP/M Plus distribution disks.

How to Start CP/M Plus

Starting or loading CP/M Plus means reading a copy of the operating system from your CP/M Plus system disk into your computer's memory.

First, check that your computer's power is on. Next, insert the CP/M Plus system disk into your drive. On CPC6128 type ¦CPM and press RETURN. On PCW 8256 simply press the space bar. This loads CP/M Plus into memory. This process is called booting, cold starting, or loading the system.

After CP/M Plus is loaded into memory, a message similar to the following is displayed on your screen:

```
   CP/M Plus      Amstrad Consumer Electronics plc

   v1.n, 61K TPA, n disk drive(s), [1 serial port |SIO/Centronics add-on],
   [112|368K drive M:]
```

The version number tells you the version of CP/M Plus that you own. After this display, the following two-character message appears on your screen:

```
A>
```

This is the CP/M Plus system prompt. The system prompt tells you that CP/M Plus is ready to read a command from your keyboard. In this example, the prompt also tells you that drive A is your default drive. This means that until you tell CP/M Plus to do otherwise, it looks for program and data files on the disk in drive A. It also tells you that you are logged in as user 0, by the absence of a user number other than 0.

The Command Line

CP/M Plus performs tasks according to specific commands that you type at your keyboard. A CP/M Plus command line is composed of a command keyword, an optional command tail, and a carriage return keystroke. The command keyword identifies a command (program) to be executed. The command tail can contain extra information for the command, such as a filename or parameters. To end the command line, you must press the carriage return, or ENTER key. The following example shows a command line.

```
A>DIR MYFILE
```

The characters that the user types are slanted to distinguish them from characters that the system displays. In this example, DIR is the command keyword and MYFILE is the command tail. The carriage return keystroke does not appear on the screen or in the example. You must remember to press the carriage return key to send a command line to CP/M Plus for processing. Note that the carriage return key is marked RETURN on your keyboard. In this guide, RETURN signifies the carriage return key.

As you type characters at the keyboard, they appear on your screen. The single-character position indicator, called the cursor, moves to the right as you type characters. If you make a typing error, press either the DELETE key or CTRL-H to move the cursor to the left and correct the error. CTRL is the abbreviation for the control key. To type a control character, hold down the Control key and press the required letter key. For example, to move the cursor to the left, hold down CTRL and press the H Key.

You can type the keyword and command tail in any combination of upper-case and lower-case letters. CP/M Plus treats all letters in the command line as upper-case.

Generally, you type a command line directly after the system prompt. However, CP/M Plus does allow spaces between the prompt and the command keyword.

CP/M Plus recognizes two different types of commands: built-in commands and transient utility commands. Built-in commands execute programs that reside in memory as a part of the CP/M Plus operating system. Built-in commands can be executed immediately. Transient utility commands are stored on disk as program files. They must be loaded from disk to perform their task. You can recognize transient utility program files when a directory is displayed on the screen because their filenames are followed by COM. Section 4 presents lists of the CP/M Plus built-in and transient utility commands.

For transient utilities, CP/M Plus checks only the command keyword. If you include a command tail, CP/M Plus passes it to the utility without checking it because many utilities require unique command tails. A command tail cannot contain more than 128 characters. Of course, CP/M Plus cannot read either the command keyword or the command tail until you press the RETURN key.

Let's use one command to demonstrate how CP/M Plus reads command lines. The DIR command, which is an abbreviation for directory, tells CP/M Plus to display a directory of disk files on your screen. Type the DIR keyword after the system prompt, omit the command tail, and press RETURN.

```
A>DIR
```

CP/M Plus responds to this command by writing the names of all the files that are stored on the disk in drive A. For example, if you have your CP/M Plus system disk in drive A, these filenames, among many others, appear on your screen:

```
ERASE    COM
PIP      COM
SET      COM
```

CP/M Plus recognizes only correctly spelled command keywords. If you make a typing error and press RETURN before correcting your mistake, CP/M Plus echoes the command line followed with a question mark. If you mistype the DIR command, as in the following example, CP/M Plus responds

```
A>DJR
DJR?
```

to tell you that it cannot find the command keyword. To correct simple typing errors, use the DELETE key, or hold down the CTRL key and press H to move the cursor to the left. CP/M Plus supports other control characters that help you efficiently edit command lines. Section 3 tells how to use control characters to edit command lines and other information you enter at your console.

DIR accepts a filename as a command tail. You can use DIR with a filename to see if a specific file is on the disk. For example, to check that the transient utility program SETSIO.COM is on your system disk, type

```
A>DIR SETSIO.COM
```

CP/M Plus performs this task by displaying either the name of the file you specified, or the message

```
No File.
```

Be sure you type at least one space after DIR to separate the command keyword from the command tail. If you do not, CP/M Plus responds as follows:

```
A>DIRSETSIO.COM
DIRSETSIO.COM?
```

Why You Should Back Up Your Files

Humans have faults, and so do computers. Human or computer errors sometimes destroy valuable programs or data files. By mistyping a command, for example, you could accidentally erase a program that you just created or a data file that has been months in the making. A similar disaster could result from an electronic component failure.

Data processing professionals avoid losing programs and data by making copies of valuable files. Always make a working copy of any new program that you purchase and save the original. If the program is accidentally erased from the working copy, you can easily restore it from the original.

It is also wise to make frequent copies of new programs or data files as you develop them. The frequency of making copies varies with each programmer. However, as a general rule, make a copy at the point where it takes ten to twenty times longer to reenter the information than it takes to make the copy.

So far, we have not discussed any commands that change information recorded on your CP/M Plus system disk. Before we do, make a few working copies of your distribution disks.

How to Make Copies of Your CP/M Plus Disks

Before using your CP/M Plus system you should make yourself a second set of system discs. Use the copies for day to day use and keep the originals in a safe place. You will need two new disks. To make the copies use the DISCKIT program (called DISCKIT3 on CPC6128) following the instructions given in your User Guide.

Section 2

Files, Disks, and Drives

CP/M Plus's most important task is to access and maintain files on your disks. With CP/M Plus you can create, read, write, copy, and erase disk files. This section tells you what a file is, how to create, name, and access a file, and how files are stored on your disks. It also tells how to change disks and change the default drive.

What is a File?

A CP/M Plus file is a collection of related information stored on a disk. Every file must have a unique name because CP/M Plus uses that name to access that file. A directory is also stored on each disk. The directory contains a list of the filenames stored on that disk and the locations of each file on the disk.

In general, there are two kinds of files: program (command) files and data files. A program file contains an executable program, a series of instructions that the computer follows step-by-step. A data file is usually a collection of information: a list of names and addresses, the inventory of a store, the accounting records of a business, the text of a document, or similar related information. For example, your computer cannot execute names and addresses, but it can execute a program that prints names and addresses on mailing labels.

A data file can also contain the source code for a program. Generally, a program source file must be processed by an assembler or compiler before it becomes a program file. In most cases, an executing program processes a data file. However, there are times when an executing program processes a program file. For example, the copy program PIP can copy one or more program files.

How Are Files Created?

There are many ways to create a file. One way is to use a text editor. The CP/M Plus text editor ED (described in Section 6) can create a file and assign it the name you specify. You can also create a file by copying an existing file to a new location, perhaps renaming it in the process. Under CP/M Plus, you can use the PIP command to copy and rename files. Finally, some programs such as MAC™ create output files as they process input files.

How Are Files Named?

CP/M Plus identifies every file by its unique file specification. A file specification can be simply a one- to eight-character filename, such as:

```
MYFILE
```

A file specification can have four parts: a drive specifier, a filename, a filetype, and a password.

The drive specifier is a single letter (A-P) followed by a colon. Each drive in your system is assigned a letter. When you include a drive specifier as part of the file specification, you are telling CP/M Plus that the file is stored on the disk currently in that drive. For example, if you enter

```
B:MYFILE
```

CP/M Plus looks in drive B for the file MYFILE.

On single disk systems you have to swap the disk in the drive or turn the disk over.

The filename can be from one to eight characters. When you make up a filename, try to let the name tell you something about what the file contains. For example, if you have a list of customer names for your business, you could name the file:

```
CUSTOMER
```

so that the name gives you some idea of what is in the file.

As you begin to use your computer with CP/M Plus, you will find that files fall naturally into categories. To help you identify files belonging to the same category, CP/M Plus allows you to add an optional one- to three-character extension, called a filetype, to the filename. When you add a filetype to the filename, separate the filetype from the filename with a period. Try to use three letters that tell something about the file's category. For example, you could add the following filetype to the file that contains a list of customer names:

```
CUSTOMER.NAM
```

When CP/M Plus displays file specifications in response to a DIR command, it adds blanks to short filenames so that you can compare filetypes quickly. The program files that CP/M Plus loads into memory from a disk have different filenames, but all have the filetype COM.

In CP/M Plus, you can add a password as an optional part of the file specification. The password can be from one to eight characters. If you include a password, separate it from the filetype (or filename, if no filetype is included) with a semicolon, as follows:

```
CUSTOMER.NAM;ACCOUNT
```

If a file has been protected with a password, you must ENTER the password as part of the file specification to access the file. Section 2.7.3 describes passwords in more detail.

We recommend that you create filenames, filetypes, and passwords from letters and numbers. You must not use the following characters in filenames, filetypes, or passwords because they have special meanings for CP/M Plus:

```
< > = , ! | * ? & / $ [ ] ( ) . : ; \ + -
```

A complete file specification containing all possible elements consists of a drive specification, a primary filename, a filetype, and a password, all separated by their appropriate delimiters, as in the following example:

```
A:DOCUMENT.LAW;SUSAN
```

Do You Have the Correct Drive?

On a two disk system when you type a file specification in a command without a drive specifier the program looks for the file in the drive named by the system prompt; this is called the default drive. For example, if you type the command

```
A>DIR MYFILE.COM
```

DIR looks in the directory of the disk in drive A for MYFILE.COM. If you have another drive, B for example, you need a way to tell CP/M Plus to access the disk in drive B instead. For this reason, CP/M Plus lets you precede a filename with a drive specifier. For example, in response to the command

```
A>DIR B:MYFILE.LIB
```

CP/M Plus looks for the file MYFILE.LIB in the directory of the disk in drive B. When you give a command to CP/M Plus, note which disk is in the default drive. Many application programs require that the data files they access be stored in the default drive.

You can also precede a program filename with a drive specifier, even if you use the program filename as a command keyword. For example, if you type the following command:

```
A>B:PIP
```

CP/M Plus looks in the directory of the disk in drive B for the file PIP.COM. If CP/M Plus finds PIP on drive B, it loads PIP into memory and executes it. If you need to access many files on the same drive, you might find it convenient to change the default drive so that you do not need to repeatedly enter a drive specifier. To change the default drive, enter the drive specifier next to the system prompt and press RETURN. In response, CP/M Plus changes the system prompt to display the new default drive:

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

Unlike the filename and filetype, which are stored in the disk directory, the drive specifier for a file changes as you move the disk from one drive to another. Therefore, a file has a different file specification when you move a disk from one drive to another. Section 4 presents more information on how CP/M Plus locates program and data files.

Do You have the Correct User Number?

CP/M Plus further identifies all files by assigning each one a user number which ranges from 0 to 15. CP/M Plus assigns the user number to a file when the file is created. User numbers allow you to separate your files into sixteen file groups. User numbers are particularly useful for organizing files on a hard disk.

When you use a CP/M Plus utility to create a file, the file is assigned to the current user number, unless you use PIP to copy the file to another user number. You can determine the current user number by looking at the system prompt.

4A>	User number 4, drive A
A>	User number 0, drive A
2B>	User number 2, drive B

The user number always precedes the drive identifier. User 0, however, is the default user number and is not displayed in the prompt.

```
A>USER 3
3A>
```

You can change both the user number and the drive by entering the new user number and drive specifier together at the system prompt:

```
A>3B:
3B>
```

Most commands can access only files that have the current user number, e.g. if the number is 7, a DIR command with no options diplays only files created under user number 7. However, if a file resides in user 0 and is marked with a special file attribute, the file can be accessed from any user number. *Note:* Locoscript, on PCW8256, holds limbo files within user numbers 8 to 15. Therefore, if you use a disk with CP/M Plus files in user 8–15, Locoscript will treat them as limbo files.

Accessing More Than One File

Certain CP/M Plus built-in and transient utilities can select and process several files when special wildcard characters are included in the filename or filetype. A file specification containing wildcards is called an ambiguous filespec and can refer to more than one file because it gives CP/M Plus a pattern to match. CP/M Plus searches the disk directory and selects any file whose filename or filetype matches the pattern.

The two wildcard characters are ?, which matches any single letter in the same position, and *, which matches any character at that position, and any other characters remaining in the filename or filetype. The following list presents the rules for using wildcards:

- A ? matches any character in a name, including a space character.
- An * must be the last, or only, character in the filename or filetype. CP/M Plus internally replaces an * with ? characters to the end of the filename or filetype.
- When the filename to match is shorter than eight characters, CP/M Plus treats the name as if it ends with spaces.
- When the filetype to match is shorter than three characters, CP/M Plus treats the filetype as if it ends with spaces.

Suppose, for example, you have a disk that contains the following six files:

A.COM AA.COM AAA.COM B.COM A.ASM and B.ASM

The following wildcard specifications match all, or a portion of, these files:

.	is treated as ????????.???
????????.???	matches all six names
*.COM	is treated as ????????.COM

????????.COM	matches the first four names
?.COM	matches A.COM and B.COM
?.*	is treated as ?.???
?.???	matches A.COM, B.COM, A.ASM, and B.ASM
A?.COM	matches A.COM and AA.COM
A*.COM	is treated as A???????.COM
A???????.COM	matches A.COM, AA.COM, and AAA.COM

Remember that CP/M Plus uses wildcard patterns only while searching a disk directory, and therefore wildcards are valid only in filenames and filetypes. You cannot use a wildcard character in a drive specifier. You also cannot use a wildcard character as part of a filename or filetype when you create a file.

How to Protect Your Files

Under CP/M Plus you can organize your files into groups to protect them from accidental change and from unauthorized access. You can specify how your files are displayed in response to a DIR command, and monitor when your files were last accessed or modified. CP/M Plus supports these features by assigning the following to files:

- user numbers
- attributes
- time and date stamps
- passwords

All of this information for each file is recorded in the disk directory.

File Attributes

File attributes control how a file can be accessed. When you create a file, CP/M Plus gives it two attributes, changeable with a SET command.

The first attribute can be set to either DIR (Directory) or SYS (System). This attribute controls whether CP/M Plus displays the file's name in response to a DIR command or DIRSYS command. When you create a file, CP/M Plus automatically sets this attribute to DIR. You can display the name of a file marked with the DIR attribute with a DIR command. If you give a file the SYS attribute, you must use a DIRSYS command to display the filename. Simple DIR and DIRSYS commands display only the filenames created under the current user number.

A file with the SYS attribute has a special advantage when it is created under user 0. When you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number. This feature gives you a convenient way to make your commonly used programs available under any user number. Note, however, that a user 0 SYS file does not appear in response to a DIRSYS command unless 0 is the current user number.

The second file attribute can be set to either R/W (Read-Write) or R/O (Read-Only). If a file is marked R/O, any attempt to write data to that file produces a Read-Only error message. Therefore, you can use the R/O attribute to protect important files. A file with the R/W attribute can be read or written to, or erased at any time, unless the disk is physically write-protected.

Date and Time Stamping

If you use date and time stamps, you can quickly locate the most recent copy of a file, and check when it was last updated or changed. You can choose to have the system tell you either when you created the file, or when you last read from or wrote to the file. You use the SET command to enable date and time stamping, and the DIR command with the DATE option to display a file's time and date stamp.

Note that Locoscript 1.20 does not update access times. Earlier versions of Locoscript cannot read disks with extended directories.

A SET command enables the option you want to monitor. You can use the following commands to enable time and date stamping on a disk, but you must choose between ACCESS and CREATE. If you choose ACCESS, the stamp records the last time the file was accessed. If you choose CREATE, the stamp records when the file was created.

```
A>SET  [ACCESS=ON]
A>SET  [CREATE=ON]
A>SET  [UPDATE=ON
```

Files created on or copied to a disk that has time and date stamping are automatically stamped. The DATE command allows you to display and reset the time and date that CP/M Plus is using. For a complete discussion of time and date stamping, see the descriptions of the SET and INITDIR commands in Section 5.

Passwords

Passwords allow you to protect your files from access by other users. You can use passwords to limit access to certain files for security purposes.

The SET utility allows you to enable password protection on a drive, assign a password to SET itself (so that unauthorized users cannot disable password protection on a drive), and assign passwords to specific files that have already been created. You can assign passwords to all program and data files. This means that a command line could require the entry of two passwords in order to execute: one password to access the command program, and a second password to access the file specified in the command tail.

Some CP/M Plus commands and most word processing, accounting, and other application programs running under CP/M Plus do not accept passwords in the command tail. For example, passwords set under CP/M Plus are ignored by Locoscript. If you want to protect your file and still use those programs, you can set a default password before executing the application program. See the description of the SET command in Section 5 for an explantion of this process.

How Are Files Stored on a Disk?

CP/M Plus records the filename, filetype, password, user number, and attributes of each file in a special area of the disk called the directory. In the directory, CP/M Plus also records which parts of the disk belong to which file.

CP/M Plus allocates directory and storage space for a file as records are added to the file. When you erase a file, CP/M Plus reclaims storage in two ways: it makes the file's directory space available to catalog a different file, and frees the file's storage space for later use. It is this dynamic allocation feature that makes CP/M Plus powerful. You do not have to tell CP/M Plus how big your file will become, because it automatically allocates more storage for a file as needed, and releases the storage for reallocation when the file is erased. Use the SHOW command to determine how much space remains on the disk.

Changing Disks

CP/M Plus cannot, of course, do anything to a file unless the disk that holds the file is inserted into a drive and the drive is ready. When a disk is in a drive, it is online and CP/M Plus can access its directory and files.

At some time, you will need to take a disk out of a drive and insert another that contains different files. You can replace an online disk whenever you see the system prompt at your console. This is a clear indication that no program is reading or writing to the drive.

You can also remove a disk and insert a new one when an application program prompts you to do so. This can occur, for example, when the data that the program uses does not fit on one disk.

Note: you must never remove a disk if a program is reading or writing to it.

You can change disks on the drive without sending any special signals to CP/M Plus. This allows you to insert another disk at a program's request and read files from or create files on the new disk.

Protecting a Drive

Under CP/M Plus, drives can be marked R/O just as files can be given the R/O attribute. The default state of a drive is R/W. You can give a drive the R/O attribute by using the SET command described in Section 5. To return the drive to R/W, use the SET command or press a CTRL-C at the system prompt.

Section 3

Console and Printer

This section describes how CP/M Plus communicates with your console and printer. It tells how to start and stop console and printer output, edit commands you enter at your console, and redirect console and printer input and output. It also explains the concept of logical devices under CP/M Plus.

Controlling Console Output

Sometimes CP/M Plus displays information on your screen too quickly for you to read it. Sometimes an especially long display scrolls off the top of your screen before you have a chance to study it. To ask the system to wait while you read the display, hold down the CONTROL (CTRL) key and press S. Note that the ALT key on PCW8256 has the same effect as CTRL. A CTRL-S keystroke causes the display to pause. When you are ready, press CTRL-Q to resume the display. Make sure you do not press any other key or key combinations during a display pause.

DIR, TYPE, and other CP/M Plus utilities support automatic paging at the console. This means that if the program's output is longer than what the screen can display at one time, the display automatically halts when the screen is filled. When this occurs, CP/M Plus prompts you to press RETURN to continue.

Controlling Printer Output

You can also use a control command to echo console output to the printer. To start printer echo, press a CTRL-P. To stop, press CTRL-P again. While printer echo is in effect, any characters that appear on your screen are listed at your printer.

You can use printer echo with a a DIR command to make a list of files stored on a floppy disk. You can also use CTRL-P with CTRL-S and CTRL-Q to make a hard copy of part of a file. Use a TYPE command to start a display of the file at the console. When the display reaches the part you need to print, press CTRL-S to stop the display, CTRL-P to enable printer echo, and then CTRL-Q to resume the display and start printing. You can use another CTRL-S, CTRL-P, CTRL-Q sequence to terminate printer echo.

Console Line Editing

You can correct simple typing errors with the DEL key. CP/M Plus also supports additional line-editing functions that you perform with control characters. You can use the control characters to edit command lines or input lines to most programs.

Line Editing in CP/M Plus

CP/M Plus allows you to edit your command line without deleting all characters. Using the line-editing control characters listed in Table 3-1, you can move the cursor left and right to insert and delete characters in the middle of a command line. You do not have to retype everything to the right of your correction. In CP/M Plus, you can press RETURN when the cursor is in any position in the command line; CP/M Plus reads the entire command line. You can also recall a command for reediting and reexecution.

In the following sample session, the user has mistyped PIP, and CP/M Plus returned an error message. The user recalls the erroneous command line by pressing CTRL-W and corrects the error, (the underbar represents the cursor):

```
A>POP A:=B:*.*_            (PIP mistyped)
POP?

A>POP A:=B:*.*_            (CTRL-W recalls the line)

A>POP A:=B:*.*             (CTRL-B to beginning of line)

A>POP A:=B:*.*             (CTRL-F to move cursor right)
```

```
A>PP_ A:=B:*.*                (CTRL-G to delete error)

A>PIP_ A:=B:*.*               (type I to correct the command name)
```

To execute the corrected command line, the user can press return even though the cursor is in the middle of the line. A return keystroke, or one of its equivalent control characters, not only executes the command, but also stores the command in a buffer so that you can recall it for editing or reexecution by pressing CTRL-W.

When you insert a character in the middle of a line, characters to the right of the cursor move to the right. If the line becomes longer than your screen is wide, characters disappear off the right side of the screen. These characters are not lost. They reappear if you delete characters from the line or if you press CTRL-E when the cursor is in the middle of the line. CTRL-E moves all characters to the right of the cursor to the next line on the screen.

Table 3-1 gives a complete list of line-editing control characters. Note that your keyboard's special keys can, in many cases, emulate the control characters. Consult Appendix D and your user guide for further details.

You probably noticed that some control characters have the same meaning. For example, the CTRL-J and CTRL-M keystrokes have the same effect as pressing the RETURN key; all three send the command line to CP/M Plus for processing. Also, CTRL-H has the same effect as pressing the DEL key. Notice that when a control character is displayed on your screen, it is preceded by an up-arrow, ↑. For example, a CTRL-C keystroke appears as ↑C on your screen.

Redirecting Input and Output

CP/M Plus's PUT command allows you to direct console or printer output to a disk file. You can use a GET command to make CP/M Plus or a utility program take console input from a disk file. The following examples illustrate some of the conveniences GET and PUT offer.

You can use a PUT command to direct console output to a disk file as well as the console. With PUT, you can create a disk file containing a directory

Table 3-1. CP/M Plus Line-editing Control Characters

Character	*Meaning*
CTRL-A	Moves the cursor one character to the left.
CTRL-B	Moves the cursor to the beginning of the command line without having any effect on the contents of the line. If the cursor is at the beginning, CTRL-B moves it to the end of the line.
CTRL-E	Forces a physical carriage return but does not send the command line to CP/M Plus. Moves the cursor to the beginning of the next line without erasing the previous input.
CTRL-F	Moves the cursor one character to the right.
CTRL-G	Deletes the character indicated by the cursor. The cursor does not move.
CTRL-H	Deletes a character and moves the cursor left one character position.
CTRL-I	Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the TAB key.
CTRL-J	Sends the command line to CP/M Plus and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a CTRL-M keystroke.
CTRL-K	Deletes to the end of the line from the cursor.
CTRL-M	Sends the command line to CP/M Plus and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a CTRL-J keystroke.
CTRL-R	Retypes the command line. Places a # at the current cursor location, moves the cursor to the next line, and retypes any partial command you typed so far.

Table 3-1 (continued)

CTRL-U	Discards all the characters in the command line, places a # at the current cursor position, and moves the cursor to the next line. However, you can use a CTRL-W to recall any characters that were to the left of the cursor when you pressed CTRL-U.
CTRL-W	Recalls and displays previously entered command line both at the operating system level and within executing programs, if the CTRL-W is the first character entered after the prompt. CTRL-J, CTRL-M, CTRL-U, and RETURN define the command line you can recall. If the command line contains characters, CTRL-W moves the cursor to the end of the command line. If you press RETURN, CP/M Plus executes the recalled command.
CTRL-X	Discards all the characters left of the cursor and moves the cursor to the beginning of the current line. CTRL-X saves any characters right of the cursor.

of all files on that disk, as follows:

```
A>PUT CONSOLE OUTPUT TO FILE DIR.PRN
Putting console output to file: DIR.PRN

A>DIR
A: FILENAME TEX : FRONT TEX : FRONT BAK : ONE    BAK : THREE TEX
A: FOUR     TEX : ONE   TEX : LINEDIT TEX : EXAMP1 TXT : TWO   BAK
A: TWO      TEX : THREE BAK : EXAMP2 TXT

A>TYPE DIR.PRN
A: FILENAME TEX : FRONT TEX : FRONT BAK : ONE    BAK : THREE TEX
A: FOUR     TEX : ONE   TEX : LINEDIT TEX : EXAMP1 TXT : TWO   BAK
A: TWO      TEX : THREE BAK : EXAMP2 TXT
```

You can use a similar PUT command to direct printer output to a disk file as well as the printer.

A GET command can direct CP/M Plus or a program to read a disk file for console input instead of the keyboard. If the file is to be read by CP/M Plus, it must contain standard CP/M Plus command lines. If the file is to be read by a utility program, it must contain input appropriate for that program. A file can contain both CP/M Plus command lines and program input if it also includes a command to start a program.

You add or omit the SYSTEM option in the GET command line to specify whether CP/M Plus or a utility program is to start reading the file, as shown in the following sample session. If you omit the SYSTEM option, the system prompt returns so that you can initiate the program that is to take input from the specified file. If you include the SYSTEM option, CP/M Plus immediately takes input from the specified file.

```
3A>TYPE PIP.DAT
B:=FRONT.TEX
B:=ONE.TEX
B:=TWO.TEX

3A>GET CONSOLE INPUT FROM FILE PIP.DAT
GETTING CONSOLE INPUT FROM FILE: PIP.DAT
3A>PIP
CP/M PLUS PIP VERSION 3.0
*B:=FRONT.TEX

*B:=ONE.TEX
*
B:=TWO.TEX

*^<CR>
3A>TYPE CCP.DAT
DIR
SHOW
DIRSYS
3A>GET CONSOLE INPUT FROM FILE CCP.DAT [SYSTEM]
GETTING CONSOLE INPUT FROM FILE: CCP.DAT
3A>DIR

A: FILENAME TEX : FRONT    TEX : FRONT    BAK : ONE      BAK : THREE    TEX
A: FOUR     TEX : ONE      TEX : LINEDIT  TEX : EXAMP1   TXT : TWO      BAK
A: TWO      TEX : EXAMP3       : EXAMP2   TXT : PIP      DAT : EXAMP4
A: THREE    BAK : EXAMP5       : CCP      DAT
```

```
3A>show
A: RW, Space:3,392k
B: RW, Space:452k

3A>NON-SYSTEM FILE(S) EXIST
```

See the descriptions of GET and PUT in Section 5 for more ways to use redirected input and output.

Assigning Logical Devices

Most CP/M Plus computer systems have a traditional console with a keyboard and screen display. Many also have letter-quality printers. If you use your computer for unusual tasks, you might want to add a different kind of character device to your system: a line printer, a teletype terminal, a modem, or even a joystick for playing games. To keep track of these physically different input and output devices, CP/M Plus associates different physical devices with logical devices. Table 3-2 gives the names of CP/M Plus logical devices. It also shows the physical devices assigned to these logical devices in the distributed CP/M Plus system.

Table 3-2. CP/M Plus Logical Devices

Logical device name	*Device type*	*Physical device assignment*
CONIN:	Console input	Keyboard
CONOUT:	Console output	Screen
AUXIN:	Auxiliary input	Null
AUXOUT:	Auxiliary output	Null
LST:	List output	Printer

On Amstrad systems you can change these assignments with a DEVICE command. You can, for example, assign AUXIN and AUXOUT to a modem so that your computer can communicate with others over the telephone.

Section 4

CP/M Plus Command Concepts

A CP/M Plus command line consists of a command keyword, an optional command tail, and a carriage return keystroke. This section describes the two kinds of programs the command keyword can identify, and tells how CP/M Plus searches for a program file on a disk. This section also tells how to execute multiple CP/M Plus commands, and how to reset the disk system.

Two Kinds of Command

A command keyword identifies a program that resides either in memory as part of CP/M Plus, or on a disk as a program file. Commands that identify programs in memory are called built-in commands. Commands that identify program files on a disk are called transient utility commands.

CP/M Plus has 6 built-in commands and over 20 transient utility commands. You can add utilities to your system by purchasing various CP/M Plus-compatible application programs. If you are an experienced programmer, you can also write your own utilities that operate with CP/M Plus.

Built-in Commands

Built-in commands are part of CP/M Plus and are always available for your use regardless of which disk you have in which drive. Built-in commands reside in memory as a part of CP/M Plus and therefore execute more quickly than the transient utilities.

Some built-in commands have options that require support from a related transient utility. The related transient has the same name as the built-in and has a filetype of COM. This type of transient utility is loaded only when a built-in command line contains options that cannot be performed by the built-in command.

If you include certain options in the command tail for a built-in command, CP/M Plus might return a .COM Required message. This means that the command tail options require support from a related transient utility and CP/M Plus could not find that program file. The following files must be accessible to support all the functions these built-ins offer: ERASE.COM, RENAME.COM, TYPE.COM, and DIR.COM.

Section 5 explains in detail the built-in commands listed in Table 4-1.

Table 4-1. Built-in Commands

Command	*Function*
DIR	Displays filenames of all files in the directory except those marked with the SYS attribute.
DIRSYS	Displays filenames of files marked with the SYS (system) attribute in the directory.
ERASE	Erases a filename from the disk directory and releases the storage space occupied by the file.
RENAME	Renames a disk file.
TYPE	Displays contents of an ASCII (TEXT) file at your screen.
USER	Changes to a different user number.

CP/M Plus allows you to abbreviate the built-in commands as follows:

DIRSYS	DIRS
ERASE	ERA
RENAME	REN
TYPE	TYP
USER	USE

Transient Utility Commands

When you enter a command keyword that identifies a transient utility, CP/M Plus loads the program file from the disk and passes it any filenames, data, or parameters you entered in the command tail. Section 5 provides the operating details for the CP/M Plus transient utilities listed in Table 4-2.

Table 4-2. Transient Utility Commands

Name	*Function*
DISCKIT	Creates a new system. Note that this is called DISCKIT3 on the CPC6128.
DATE	Sets or displays the date and time.
DEVICE	Assigns logical CP/M devices to one or more physical devices, changes device driver protocol and baud rates, or sets console screen size.
DUMP	Displays a file in ASCII and hexadecimal format.
ED	Creates and alters character files.
GET	Temporarily gets console input from a disk file rather than the keyboard.
HELP	Displays information on how to use CP/M Plus commands.
HEXCOM	Uses the output from MAC to produce a program file.
INITDIR	Initializes a disk directory to allow time and date stamping.
LANGUAGE	Selects character set.
LINK	Links REL (relocatable) program modules produced by RMAC (relocatable macro assembler) and produces program files.

Table 4-2 (continued)

MAC	Translates assembly language programs into machine code form.
PALETTE	Sets ink colours.
PAPER	Initializes FX-80 or PCW8256 printer (8256 only).
PIP	Copies files and combines files.
PUT	Temporarily directs printer or console output to a disk file.
RMAC	Translates assembly language programs into relocatable program modules.
SET	Sets file options including disk labels, file attributes, type of time and date stamping, and password protection.
SETDEF	Sets system options including the drive search chain.
SETKEYS	Configures the keyboard.
SETLST	Performs printer initialization.
SET24X80	Sets the screen into 24x80 mode.
SHOW	Displays disk and drive statistics.
SID	Helps you check your programs and interactively correct programming errors.
SUBMIT	Automatically executes multiple commands.
XREF	Produces a cross-reference list of variables used in an assembler program.

How CP/M Plus Searches for Program and Data Files

This section describes how CP/M Plus searches for program and data files on disk. If it appears that CP/M Plus cannot find a program file you specified in a command line, the problem might be that CP/M Plus is not looking on the drive where the file is stored. Therefore, you need to understand the steps CP/M Plus follows in searching for program and data files.

Finding Data Files

When you enter a command line, CP/M Plus passes the command tail to the program identified by the command keyword. If the command tail contains a file specification, the program calls CP/M Plus to search for the data file. If CP/M Plus cannot find the data file, the program displays an error message at the console. Typically, this message is "File not found" or "No File," but the message depends on the program identified by the command keyword.

If you do not include a drive specifier with the filename in a command tail, CP/M Plus searches the directory of the current user number on the default drive. If the file is not there, CP/M Plus looks for the file with the SYS attribute in the directory of user 0 on the default drive. If CP/M Plus finds the file under user 0, it allows the program Read-Only access to the file. For example, if you enter the following command line,

```
3A>TYPE MYFILE.TXT
```

CP/M Plus first searches the directory for user 3 on drive A. If it does not find MYFILE.TXT there, it searches the directory of user 0 on drive A for MYFILE.TXT marked with the SYS attribute. If the file is not in either directory, CP/M Plus returns control to TYPE, which then displays "No File."

Some CP/M Plus utilities, such as PIP and DIR, restrict their file search to the current user number. Because CP/M Plus does not allow Read-Write access to SYS files, ERASE and RENAME also restrict their search to the current user number.

The search procedure is basically the same if you do include a drive specifier with the filename. CP/M Plus first looks in the directory of the current user number on the specified drive. Then, if it does not find the file, it looks in the directory for user 0 on the specified drive for the file with the SYS attribute. If CP/M Plus does not find the data file after these two searches, it displays an error message.

Finding Program Files

The search procedure for a program file can be very different from a data file search. This is because you can use the SETDEF command described in Section 5 to define the search procedure you want CP/M Plus to follow when it is looking for a program file. With SETDEF you can ask CP/M Plus to make as many as sixteen searches when you do not include a drive specifier before the command keyword, but that is a rare case! We will begin by describing how CP/M Plus searches for program files when you have not yet entered a SETDEF command.

If a command keyword identifies a transient utility, CP/M Plus looks for that program file on the default or specified drive. It looks under the current user number, and then under user 0 for the same file marked with the SYS attribute. At any point in the search process, CP/M Plus stops the search if it finds the program file. CP/M Plus then loads the program into memory and executes it. When the program terminates, CP/M Plus displays the system prompt and waits for your next command. However, if CP/M Plus does not find the command file, it repeats the command line followed by a question mark, and waits for your next command.

If you include a drive specifier before the command keyword, you are telling CP/M Plus precisely where to look for the program file. Therefore, CP/M Plus searches only two locations: the directory for the current user on the specified drive, and then for user 0 on the specified drive, before it repeats the command line with a question mark. For example, if you enter

```
4B>A:SHOW [SPACE]
```

CP/M Plus looks on drive A, user 4 and then user 0 for the file SHOW.COM.

If you do not include a drive specifier before the command keyword, CP/M Plus searches directories in a sequence called a drive chain. When you first receive CP/M Plus, there is only one drive in your chain, the default drive. Unless you change the chain with a SETDEF command, CP/M Plus looks in two places for the program file. For example, if you enter

```
7B>SHOW [SPACE]
```

CP/M Plus searches the following locations for the file SHOW.COM:

1 drive B, user 7

2 drive B, user 0

Remember that a SHOW.COM file under user 0 must be marked with the SYS attribute or else CP/M Plus cannot find it. Use a SET command to give program files under user 0 to the SYS attribute because they can then be accessed automatically from all other user areas. You do not have to duplicate frequently used program files in all user areas on all drives.

When you use a SETDEF command to define your own drive chain, include the default drive, and the drive that contains your most frequently used utilities. For an example, assume you defined your drive chain as * (the default drive) and drive A. When you enter the following command:

```
2B>SHOW [SPACE]
```

CP/M Plus looks for SHOW.COM in the following sequence:

1 drive B, user 2

2 drive B, user 0

3 drive A, user 2

4 drive A, user 0

You can include your default drive in your drive chain with an option in a SETDEF command. Any drive chain you specify with SETDEF remains in effect until you restart or reset the system.

You can also use a SETDEF command to enable automatic submit in your drive chain.

Executing Multiple Commands

In the examples so far, CP/M Plus has executed only one command at a time. CP/M Plus can also execute a sequence of commands. You can enter a sequence of commands at the system prompt, or you can put a frequently needed sequence of commands in a disk file. Once you have stored the sequence in a disk file, you can execute the sequence whenever you need to with a SUBMIT command.

To enter multiple commands at the system prompt, separate each command keyword and associated command tail from the next keyword with an exclamation point, !. When you complete the sequence, press RETURN. CP/M Plus executes your commands in order:

```
3A>DIRSYS!DIR EXAMP*.*!SHOW [SPACE]
NON-SYSTEM FILE(S) EXIST
3A>DIR EXAMP*.*
A: EXAMP7   : EXAMP1  TXT : EXAMP3   : EXAMP2  TXT : EXAMP4
A: EXAMP5   : EXAMP6
3A>SHOW [SPACE]

A: RW, SPACE:   44K
```

If you find you need to execute the same command sequence frequently, store the sequence in a disk file. To create this file, use ED or another character file editor. The file must have a filetype of SUB. Each command in the file must start on a new line. For example, an UPDATE.SUB file might look like this:

```
DIR A:*.COM
ERA B:*.COM
PIP B:=A:*.COM
```

To execute this list, enter the following command:

```
A>SUBMIT UPDATE
```

The SUBMIT utility passes each command to CP/M Plus for sequential execution. While SUBMIT executes, the commands are usually echoed at the console, as well as any program's screen display, such as the directory or PIP's "COPYING..." message. When one command completes, the system prompt reappears either with the next command in the SUB file, or, when the SUB file is exhausted, by itself to wait for your next command from the keyboard.

PROFILE.SUB is a special submit file that CP/M Plus automatically executes at each cold start. This feature is especially convenient if you regularly execute a standard set of commands, such as SETDEF and DATE SET, before beginning a work session.

The description of the SUBMIT utility in Section 5 gives more details on how to create a SUB file and use SUBMIT parameters to pass options to the programs to be executed.

You can also use CTRL-C to reset the disk system. This is sometimes called a warm start. When you press CTRL-C and the cursor is at the system prompt, CP/M Plus logs out all the active drives, then logs in the default drive. The active drives are any drives you have accessed since the last cold or warm start. A SHOW [SPACE] command displays the remaining space on all active drives. In the example in Terminating Programs (below), SHOW [SPACE] indicates that three drives are active. However, if you press CTRL-C immediately after this display and then enter another SHOW [SPACE] command, only the space for the default drive, A, is displayed.

Terminating Programs

You can use the two keystroke command CTRL-C (ALT-C on PCW8256) to terminate program execution or reset the disk system. To enter a CTRL-C command, hold down the CTRL key and press C.

Not all application programs can be terminated by a CTRL-C. However, most of the transient utilities supplied with CP/M Plus can be terminated immediately by a CTRL-C keystroke. If you try to terminate a program while it is sending a display to the screen, you might need to press a CTRL-S to halt the display before entering CTRL-C.

You can also use CTRL-C to reset the disk system. This is sometimes called a warm start. When you press CTRL-C and the cursor is at the system prompt, CP/M Plus logs out all the active drives, then logs in the default drive. The active drives are any drives you have accessed since the

last cold or warm start. A SHOW [SPACE] command displays the remaining space on all active drives. In the following example, SHOW [SPACE] indicates that three drives are active. However, if you press CTRL-C immediately after this display and then enter another SHOW [SPACE] command, only the space for the default drive, A, is displayed.

```
A>SHOW [SPACE]
A:  RW,  Space:      88k
B:  RO,  Space:      54k
M:  RO,  Space:      65k
A>^C
A>SHOW [SPACE]
A:  RW,  Space:      88k
```

Getting Help

CP/M Plus includes a transient utility command called HELP that can display a summary of what you need to know to use each command described in this manual. To get help, simply enter the command:

```
A>HELP
```

In response, the HELP utility displays a list of topics for which summaries are available. After HELP lists the topics available, it displays its own prompt:

```
HELP>
```

To this prompt, you can enter one of the topics presented in the list, for example,

```
HELP>SHOW
```

After displaying a summary of the SHOW command, HELP lists subtopics that detail different aspects of the SHOW command. To display the information on a subtopic when you have just finished reading the main topic, enter the name of the subtopic preceded by a period:

```
HELP>.OPTIONS
```

In the preceding example, HELP then displays the options available for the SHOW command. As you become familiar with HELP, you might want to call a HELP subtopic directly from the system prompt as follows:

```
A>HELP SHOW OPTIONS
```

HELP lets you learn the basic CP/M Plus commands quickly. You might find that you reference the command summary in Section 5 only when you need details not provided in the HELP summaries. When you add new utilities, you can modify HELP to add or subtract topics, or even modify the summaries HELP presents. See the description of HELP in Section 5 for complete details.

Section 5

Command Summary

This section describes the commands and programs supplied with your CP/M Plus operating system. The commands are in alphabetical order. Each command is followed by a short explanation of its operation and examples. More complicated commands are described later in detail. For example, ED is described in Section 6.

CP/M Plus has replaced some commands from previous CP/M versions such as CP/M 2.2. MAC replaces ASM; SHOW and DIR include the previous STAT functions; and SID replaces DDT.

Let's Get Past the Formalities

This section describes the parts of a file specification in a command line. There are four parts in a file specification; to avoid confusion, each part has a formal name:

- drive specifier — the optional disk drive A, B, M that contains the file or group of files to which you are referring. If a drive specifier is included in your command line, a colon must follow it.

- filename — the one- to eight-character first name of a file or group of files.

- filetype — the optional one- to three-character category name of a file or group of files. If the filetype is present, a period must separate it from the filename.

- password — the optional one- to eight-character password which allows you to protect your files. It follows the filetype, or the filename if no filetype is assigned, and is preceded by a semicolon.

On twin disk systems,if you do not include a drive specifier, CP/M Plus automatically uses the default drive. If you omit the period and the filetype, CP/M Plus automatically includes a filetype of three blanks.

This general form is called a file specification. A file specification names a particular file or group of files in the directory of the on-line disk given by the drive specifier. For example,

```
B:MYFILE.DAT
```

is a file specification that indicates drive B:, filename MYFILE, and filetype DAT. File specification is abbreviated to

  filespec

in the command syntax statements.

Some CP/M Plus commands accept wildcards in the filename and filetype parts of the command tail. For example,

```
B:MY*.A??
```

is a file specification with drive specifier B:, filename MY*, and filetype A??. This ambiguous file specification might match several files in the directory.

Put together, the parts of a file specification are represented like this:

  d:filename.typ;password

In the preceding form, d: represents the optional drive specifier, filename represents the one- to eight-character filename, and typ represents the optional one- to three-character filetype. The syntax descriptions in this section use the term filespec to indicate any valid combination of the elements included in the file specification. The following list shows valid combinations of the elements of a CP/M Plus file specification.

- filename
- filename.typ
- filename;password

- filename.typ;password
- d:filename
- d:filename.typ
- d:filename;password
- d:filename.typ;password

Table 5-1. Reserved Characters

Character	*Meaning*
tab space carriage return	file specification delimiters
:	drive delimiter in file specification
.	filetype delimiter in file specification
;	password delimiter in file specification
* ?	wildcard characters in an ambiguous file specification
< > & ! | / + -	option list delimiters
[ ]	option list delimiters for global and local options
( )	delimiters for multiple modifiers inside square brackets for options that have modifiers
/ $	option delimiters in a command line
;	comment delimiter at the beginning of a command line

The characters in Table 5-1 have special meaning in CP/M Plus, so do not use these characters in file specifications except as indicated.

CP/M Plus has already established several file groups. Table 5-2 lists some of their filetypes with a short description of each family. Appendix C provides the complete list.

Table 5-2. CP/M Plus Filetypes

Filetype	*Meaning*
ASM	Assembler source file
BAS	BASIC source program
COM	Machine language program
HLP	HELP message file
SUB	List of commands to be executed by SUBMIT
$$$	Temporary file

In some commands, descriptive qualifiers are used with filespecs to further qualify the type of filespec accepted by the commands. For example, wildcard-filespec denotes wildcard specifications, dest-filespec denotes a destination filespec, and src-filespec denotes a source filespec.

You now understand command keywords, command tails, control characters, default drive, and wildcards. You also see how to use the formal names filespec, drive specifier, filename, and filetype. These concepts give you the background necessary to compose complete command lines.

How Commands Are Described

CP/M Plus commands appear in alphabetical order. Each command description is given in a specific form. This section also describes the notation that indicates the optional parts of a command line and other syntax notation:

- The description begins with the command keyword in upper-case.

- The syntax section gives you one or more general forms to follow when you compose the command line.

- The explanation section defines the general use of the command keyword, and points out exceptions and special cases. The explanation sometimes includes tables or lists of options that you can use in the command line.

- The examples section lists a number of valid command lines that use the command keyword. To clarify examples of interactions between you and the operating system, the characters that you enter are slanted. The responses that CP/M Plus shows on your screen are in vertical type.

The notation in the syntax lines describes the general command form using these rules:

- Words in capital letters must be spelled as shown, but you can use any combination of upper- or lower-case letters.

- The symbolic notation d:, filename, .typ, ;password, and filespec have the general meanings described on pages 55–56.

- You must include one or more space characters where a space is shown, unless otherwise specified. For example, the PIP options do not need to be separated by spaces.

The following table defines the special symbols and abbreviations used in syntax lines.

Table 5-3. Syntax Notation

Symbol	*Meaning*
DIR	Directory attribute.
n	You can substitute a number for n.
o	Indicates an option or an option list.
RO	Read-Only.
RW	Read-Write.
s	You can substitute a string, which consists of a group of characters, for s.
SYS	System attribute.
{}	Items within braces are optional. You can enter a command without the optional items. The optional items add effects to your command line.
[]	Items in square brackets are options or an option list. If you use an option specified within the brackets, you must type the brackets to enclose the option. If the right bracket is the last character on the command line, it can be omitted.
()	Items in parentheses indicate a range of options. If you use a range from an option list, you must enclose the range in parentheses.
...	Ellipses tell you that the previous item can be repeated any number of times.
|	The bar separates alternative items in a command line. You can select any or all of the alternatives specified. Mutually exclusive options are indicated in additional syntax lines or are specifically noted in the text.

Table 5-3 (continued)

↑ or CTRL	Represent the CTRL key on your keyboard. (CTRL characters are prefixed ↑ on your screen.)
<cr>	Indicates a carriage return keystroke.
*	Wildcard character — replaces all or part of a filename and/or filetype.
?	Wildcard character — replaces any single character in the same position of a filename or filetype.

Let's look at some examples of syntax notation. The CP/M Plus DIR (DIRectory) command displays the names of files cataloged in the disk directory and, optionally, displays other information about the files.

The syntax of the DIR command with options shows how to use the command line syntax notation:

```
Syntax:    DIR {d:} | {filespec}{[options]}
                 |       |        023
           optional optional optional
```

This tells you that the command tail following the command keyword DIR is optional. DIR alone is a valid command, but you can include a file specification, or a drive specification, or just the options in the command line. Therefore,

```
DIR
DIR filespec
DIR d:
DIR [RO]
```

are valid commands. Furthermore, the drive or file specification can be followed by another optional value selected from the following list of DIR options:

```
RO
RW
DIR
SYS
```

Therefore,

```
DIR d:filespec [RO]
```

is a valid command.

Recall that in Section 2 you learned about wildcards in filenames and filetypes. The DIR command accepts wildcards in the file specification.

Using this syntax, you can construct several valid command lines:

```
DIR
DIR X.PAS
DIR X.PAS [RO]
DIR X.PAS [SYS]
DIR *.PAS
DIR *.* [RW]
DIR X.* [DIR]
```

The CP/M Plus command PIP (Peripheral Interchange Program) is the file copy program. PIP can copy information from the disk to the screen or printer. PIP can combine two or more files into one longer file. PIP can also rename files after copying them. Look at one of the formats of the PIP command line for another example of how to use command line notation. PIP also copies files from one disk to another disk.

```
Syntax:   PIP dest-filespec=src-filespec{,filespec...}
```

In the preceding example, dest-filespec is further defined as a destination file specification or peripheral device (printer, for example) that receives data. Similarly, src-filespec is a source file specification or peripheral device (keyboard, for example) that transmits data. PIP accepts wildcards in the filename and filetype. (See the PIP command for details regarding other capabilities of PIP.) There are, of course, many valid command lines that come from this syntax. Some examples follow.

```
PIP NEWFILE.DAT=OLDFILE.DAT
PIP B:=A:THISFILE.DAT
PIP B:X.BAS=Y.BAS, Z.BAS
PIP X.BAS=A.BAS, B.BAS, C.BAS
PIP B:=A:*.BAK
PIP B:=A:*.*
```

The remainder of this section contains a complete description of each CP/M Plus utility. The descriptions are arranged alphabetically for easy reference.

The DATE Command

Syntax: DATE {CONTINUOUS}
DATE {time-specification}
DATE SET

Explanation: The DATE command is a transient utility that lets you display and set the date and time of day. When you start CP/M Plus, the date and time are set to the creation date of your CP/M Plus system. Use DATE to change this initial value to the current date and time.

Display Current Date and Time

Syntax: DATE {CONTINUOUS}

Explanation: The preceding form of the DATE command displays the current date and time. The CONTINUOUS option allows continuous display of the date and time. The CONTINUOUS option can be abbreviated to C. You can stop the continuous display by pressing any key.

Examples:

```
A>DATE
A>DATE C
```

The first example displays the current date and time. A sample display might

```
Fri 08/13/82 09:15:37
```

The second example displays the date and time continuously until you press any key to stop the display.

Set the Date and Time

Syntax: DATE {time-specification}
DATE SET

Explanation: The first form allows the user to enter both date and time in the command. The time-specification format is

MM/DD/YY HH:MM:SS

where:

MM is a month value in the range 1 to 12.
DD is a day value in the range 1 to 31.
YY is the two-digit year value relative to 1900.
HH is the hour value in the range of 0 to 23.
MM is the minute value in the range of 0 to 59.
SS is the second value in the range of 0 to 59.

The system checks the validity of the date and time entry and determines the day for the date entered.

The second form prompts you to enter the date and the time. To keep the current system date or time, press the carriage return.

Examples:

```
A>DATE 08/14/82 10:30:00
```

The system responds with

```
Press any key to set time
```

When the time occurs, press any key. DATE initializes the time at that instant, and displays the date and time:

```
Sat 08/14/82 10:30:00
```

```
A>DATE SET
```

The system prompts with

```
Enter today's date (MM/DD/YY):
```

Press the carriage return to skip or enter the date. Then the system prompts with

```
Enter the time (HH:MM:SS):
```

Press the carriage return to skip or enter the time and the system prompts with

```
Press any key to set time
```

to allow you to set the time exactly.

The DEVICE Command

Syntax:

```
DEVICE {NAMES | VALUES | physical-dev | logical-dev}
DEVICE logical-dev=physical-dev {option}
                        {,physical-dev { option},...}
DEVICE logical-dev = NULL
DEVICE physical-dev {option}
DEVICE CONSOLE [PAGE | COLUMNS=columns |
LINES=lines}
```

Explanation: The DEVICE command is a transient utility that displays current assignments of CP/M Plus logical devices and the names of physical devices. DEVICE allows you to assign logical CP/M Plus devices to peripheral devices attached to the computer. The DEVICE command also sets the communications protocol and speed of a peripheral device, and displays or sets the current console screen size.

CP/M Plus supports the following five logical devices:

CONIN:
CONOUT:
AUXIN:
AUXOUT:
LST:

These logical devices are also known by the following names:

CON: (for CONIN: and CONOUT:)
CONSOLE: (for CONIN: and CONOUT:)
KEYBOARD (for CONIN:)
AUX: (for AUXIN: and AUXOUT:)
AUXILIARY: (for AUXIN: and AUXOUT:)
PRINTER (for LST:)

The physical device names on a computer vary from system to system. You can use the DEVICE command to display the names and attributes of the physical devices that your system accepts.

Display Device Characteristics and Assignments

Syntax: DEVICE { NAMES | VALUES | physical-dev | logical-dev}

Explanation: The preceding form of the DEVICE command displays the names and attributes of the physical devices and the current assignments of the logical devices in the system.

Examples:

```
A>DEVICE
```

The preceding command displays the physical devices and current assignments of the logical devices in the system. The following is a sample response:

```
Physical Devices:
I=Input,O=Output,S=Serial,X=Xon-Xoff
CRT    NONE  IO     LPT    NONE  O     IOS

Current Assignments:
CONIN:          = CRT
CONOUT:         = CRT
AUXIN:          = Null Device
AUXOUT:         = Null Device
LST:            = LPT

Enter new assignment or hit RETURN:
```

The system prompts for a new device assignment. You can enter any valid device assignment (as described in the next section). If you do not want to change any device assignments, press the RETURN key.

```
A>DEVICE NAMES
```

The preceding command lists the physical devices with a summary of the device characteristics.

```
A>DEVICE VALUES
```

The preceding command displays the current logical device assignments.

```
A>DEVICE CRT
```

The preceding command displays the attributes of the physical device CRT.

```
A>DEVICE CON
```

The preceding command displays the assignment of the logical device CON:

Assign a Logical Device

Syntax:
```
DEVICE logical-dev = physical-dev {option}
                     {,physical-dev {option},...}
DEVICE logical-dev = NULL
```

Explanation: The first form assigns a logical device to one or more physical devices. The second form disconnects the logical device from any physical device.

Table 5-4. DEVICE Options

Option	*Meaning*
XON	refers to the XON/XOFF communications protocol. This protocol uses two special characters in the ASCII character set called XON and XOFF. XON signals transmission on, and XOFF signals transmission off. Before each character is output from the computer to the peripheral device, the computer checks to see if there is any incoming data from the peripheral. If the incoming character is XOFF, the computer suspends all further output until it receives an XON from the device, indicating that the device is again ready to receive more data.
NOXON	indicates no protocol and the computer sends data to the device whether or not the device is ready to receive it.
baud-rate	is the speed of the device. The system accepts the following baud rates:
	50 75 110 134
	150 300 600 1200
	1800 2400 3600 4800
	7200 9600 19200

Examples:

```
A>DEVICE CONOUT:=LPT,CRT
A>DEVICE AUXIN:=CRT2 [XON,9600]
A>DEVICE LST:=NULL
```

The first example assigns the system console output, CONOUT:, to the printer, LPT, and the screen, CRT. The second example assigns the auxiliary logical input device, AUXIN:, to the physical device CRT using protocol XON/XOFF and sets the transmission rate for the device at 9600. The third example disconnects the list output logical device, LST:.

Set Attributes of a Physical Device

Syntax: DEVICE physical-dev {option}

Explanation: The preceding form of the DEVICE command sets the attributes of the physical device specified in the command.

Example:

```
A>DEVICE LPT [XON,9600]
```

The preceding command sets the XON/XOFF protocol for the physical device LPT and sets the transmission speed at 9600.

Display or Set the Current Console Screen Size

Syntax: DEVICE CONSOLE [PAGE | COLUMNS=columns | LINES=lines]

Explanation: The preceding form of the DEVICE command displays or sets the current console size.

Examples:

```
A>DEVICE CONSOLE [PAGE]
A>DEVICE CONSOLE [COLUMNS=40, LINES=16]
```

The first example displays the current console page width in columns and length in lines. The second example sets the screen size to 40 columns and 16 lines.

The DIR Command

Syntax:	DIR {d:}
	DIR {filespec}
	DIRSYS {d:}
	DIRSYS {filespec}
	
	DIR {d:} [options]
	DIR {filespec} {filespec}...[options]

Explanation:	The DIR command displays the names of files and the attributes associated with the files. DIR and DIRSYS are built-in utilities; DIR with options is a transient utility.

Display Directory

Syntax:	DIR {d:}
	DIR {filespec}
	
	DIRSYS {d:}
	DIRSYS {filespec}

Explanation:	The DIR and DIRSYS commands display the names of files cataloged in the directory of an on-line disk. The DIR command lists the names of files in the current user number that have the Directory (DIR) attribute. DIR accepts wildcards in the file specification. You can abbreviate the DIRSYS command to DIRS.

The DIRSYS command displays the names of files in the current user number that have the System (SYS) attribute. Although you can read System (SYS) files that are stored in user 0 from any other user number on the same drive, DIRSYS only displays user 0 files if the current user number is 0. DIRSYS accepts wildcards in the file specification.

If you omit the drive and file specifications, the DIR command displays the names of all files with the DIR attribute on the default drive for the current user number. Similarly, DIRSYS displays all the SYS files.

If the drive specifier is included, but the filename and filetype are omitted, the DIR command displays the names of all DIR files in the current user on the disk in the specified drive. DIRSYS displays the SYS files.

If the file specification contains wildcard characters, all filenames that satisfy the match are displayed on the screen.

If no filenames match the file specification, or if no files are cataloged in the directory of the disk in the named drive, the DIR or DIRSYS command displays the message:

```
No File
```

If system (SYS) files match the file specification, DIR displays the message:

```
SYSTEM FILE(S) EXIST
```

If nonsystem (DIR) files match the file specification, DIRSYS displays the message:

```
NON-SYSTEM FILES(S) EXIST
```

The DIR command pauses after filling the screen. Press any key to continue the display.

Note: You can use the DEVICE command to change the number of columns displayed by DIR or DIRSYS.

Examples:

```
A>DIR
```

Displays all DIR files cataloged in user 0 on the default drive A.

```
A>DIR B:
```

Displays all DIR files for user 0 on drive B.

```
A>DIR B:X.BAS
```

Displays the name X.BAS if the file X.BAS is present on drive B.

```
4A>DIR *.BAS
```

Displays all DIR files with filetype BAS for user 4 on drive A.

```
B>DIR A:X*.C?D
```

Displays all DIR files for user 0 on drive A whose filename begins with the letter X, and whose three character filetype contains the first character C and last character D.

```
A>DIRSYS
```

Displays all files for user 0 on drive A that have the system (SYS) attribute.

```
3A>DIRS *.COM
```

This abbreviated form of the DIRSYS command displays all SYS files with filetype COM on default drive A for user 3.

Display Directory with Options

Syntax: DIR {d:} [options] DIR {filespec} {filespec}...[options]

Explanation: The DIR command with options is an enhanced version of the DIR command. The DIR command displays CP/M Plus files in a variety of ways. DIR can search for files on any or all drives, for any or all user numbers.

DIR allows the option list to occur anywhere in the command tail. These options modify the entire command line. Only one option list is allowed.

Options must be enclosed in square brackets. The options can be used individually, or strung together separated by commas or spaces. Options can be abbreviated to only one or two letters if the abbreviation unambiguously identifies the option.

If a directory listing exceeds the size of your screen, DIR automatically halts the display when it fills the screen. Press any key to continue the display.

Table 5-5. DIR Display Options

Option	*Function*
ATT	displays the user-definable file attributes F1, F2, F3, and F4.
DATE	displays files with date and time stamps. If date and time stamping is not active, DIR displays the message: Date and Time Stamping Inactive.
DIR	displays only files that have the DIR attribute.
DRIVE=ALL	displays files on all accessed drives. DISK is also acceptable in place of DRIVE in all the DRIVE options.
DRIVE=(A,B,M)	displays files on the drives specified.

DRIVE=d

displays files on the drive specified by d.

EXCLUDE

displays the files on the default drive and user area that do not match the files specified in the command line.

FF

sends an initial form-feed to the printer device if the printer has been activated by CTRL-P. If the LENGTH=n option is also specified, DIR issues a form-feed every n lines. Otherwise, the FF option deactivates the default paged output display.

FULL

shows the name of the file and the size of the file. The size is shown as the amount of space in kilobytes and the number of 128-byte records allocated to the file. FULL also shows the attributes the file. (See the SET command for description of file attributes). If there is a directory label on the drive, DIR shows the password protection mode and the time stamps. The display is alphabetically sorted. FULL is the default output format for display when using DIR with options.

LENGTH=n

displays n lines of output before inserting a table heading. n must be in the range between 5 and 65536. The default length is one full screen of information.

Table 5-5 (continued)

MESSAGE	displays the names of the specified drives and user numbers it is currently searching. If there are no files in the specified locations, DIR displays the file not found message.
NOPAGE	continuously scrolls information by on the screen. Does not wait for you to press a key to restart the scrolling movement.
NOSORT	displays files in the order it finds them on the disk. If this option is not included, DIR displays the files alphabetically.
RO	displays only the files that have the Read-Only attribute.
RW	displays only the files that are set to Read-Write.
SIZE	displays the filename and file size in kilobytes.
SYS	displays only the files that have the SYS attribute.

USER=ALL
: displays all files under all the user numbers for the default drive.

USER=n
: displays the files under the user number specified by n.

USER=(0,1,...,15)
: displays files under the user numbers specified.

Examples:

```
A>DIR B: [FULL]
A>DIR B: [SIZE]
```

The following is sample output of the [FULL] option display format shown in the first example of the DIR command:

```
                   Directory for Drive B:   User  0

   Name              Bytes    Recs   Attributes      Prot           Update                Access
                   ------------------------------------------------------------------------------------
DITS       BAK          1k       1   Dir RW          Read    09/01/82 13:04        09/01/82 13:07
DITS       TES          1k       1   Dir RO          None    09/01/82 13:07        09/01/82 13:09
DITS       Y            1k       1   Dir RW          None    08/25/82 03:33        08/25/82 03:33
DITS       ZZ           1k       1   Dir RW          None    08/25/82 03:36        08/25/82 03:36
SETDEF     COM          4k      29   Dir RO          None                          08/25/82 03:36
SUBMIT     TX2          1k       1   Dir RO          None
SUBMIT     TX1          5k      43   Dir RO          None

Total Bytes          =      14k      Total Records = 77 Files Found = 7
Total 1k Blocks      =      14       Used/Max Dir Entries for Drive B: 11/  64
```

The following is sample output of the [SIZE] option display format shown in the second example of the DIR command:

```
              Directory for Drive B:   User 0

B: DITS       BAK      1k  :  DITS      TES     1k  :  DITS         Y        1k
B: DITS       ZZ       1k  :  SETDEF COM        4k  :  SUBMIT       TX2      1k
B: SUBMIT     TX1      5k  :

              Total Bytes = 14k Total Records = 77 Files Found = 7
              Total 1k Blocks = 14 Used/Max Dir Entries for Drive B: 11/ 64
```

Both the full format and the size format follow their display with two lines of totals. The first line displays the total number of kilobytes, the total number of records, and the total number of files for that drive and user area. The second line displays the total number of 1K blocks needed to store the listed files. The number of 1K blocks shows the amount of storage needed to store the files on a single density disk, or on any drive that has a block size of one kilobyte. The second line also shows the number of directory entries used per number of directory entries available on the drive.

```
A>DIR [DRIVE=B,FF]
```

DIR sends a form-feed to the printer before displaying the files on drive B.

```
A>DIR B: [RW,SYS]
```

The preceding example displays all the files on drive B with Read-Write and SYS attributes.

```
A>DIR B: [USER=ALL]
```

Displays all the files under each user number (0-15) on drive B.

```
A>DIR [USER=2]
```

Displays all the files under user 2 on the default drive.

```
A>DIR B: [USER=(3,4,10)]
```

This example displays all the files under user numbers 3, 4, and 10 on drive B.

```
A>DIR [DRIVE=ALL]
```

Displays all the files under user 0 on all the drives in the drive search chain. (See the SETDEF command.)

```
4A>DIR [DRIVE=B]
```

Displays all the files under user 4 on drive B.

```
A>DIR [DRIVE=(B,M)]
```

Displays all the files under user 0 on drives B and M.

```
A>DIR [exclude] *.COM
```

The preceding example above lists all the files on the default drive and user 0 that do not have a filetype of COM.

```
A>DIR [user=all,drive=all,sys] *.PLI *.COM *.ASM
```

The preceding command line instructs DIR to list all the system files of type PLI, COM, and ASM on the system in the currently active drives for all the user numbers on the drives.

```
A>DIR X.SUB [MESSAGE,USER=ALL,DRIVE=ALL]
```

The preceding command searches all drives under each user number for X.SUB. During the search, DIR displays the drives and user numbers.

```
A>DIR [drive=all user=all] TESTFILE.BOB
```

The preceding example instructs DIR to display the filename TESTFILE.BOB if it is found on any logged-in drive for any user number.

```
A>DIR [size,rw] B:
```

The preceding example instructs DIR to list each Read-Write file that resides on drive B with its size in kilobytes. Note that B: is equivalent to B:\*.*.

The DISCKIT Command

Syntax: DISCKIT (PCW8256)
DISCKIT3 (CPC6128)

Explanation: DISCKIT enables you to format your disks and copy the contents of your issued system disks by making selections from a menu. Remember that formatting a disk erases any information already held on that disk. You can make working copies of your system disks and use the ERASE option to remove any files you do not require. See your User Guide for full details.

Example:

```
DISC KIT 1.0
CPC6128 & CP/M Plus
© 1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.

one drive found

Copy                 | 7 |
Format               | 4 |
Verify               | 1 |
Exit from program    | 0 |
```

The DUMP Command

Syntax: DUMP filespec

Explanation: Dump displays the contents of a file in hexadecimal and ASCII format.

Example:

```
A>DUMP ABC.TEX
```

Console output can look like the following:

```
DUMP - Version 3.0
0000: 41 42 43 0D 0A 44 45 46 0D 0A 47 48 49 0D 0A 1A ABC..DEF..GHI...
0010: 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
```

The ED Command

Syntax: ED {input-filespec {B: | output-filespec}}

Explanation: The ED transient utility lets you create and edit a disk file.

The ED utility is a line-oriented context editor. This means that you create and change character files line-by-line, or by referencing individual characters within a line.

The ED utility lets you create or alter the file named in the file specification. Refer to Section 6 for a description of the ED utility.

The ED utility uses a portion of your user memory as the active text buffer where you add, delete, or alter the characters in the file. You use the A command to read all or a portion of the file into the buffer. You use the W or E command to write all or a portion of the characters from the buffer back to the file.

An imaginary character pointer, called CP, is at the beginning of the buffer, between two characters in the buffer, or at the end of the buffer.

You interact with the ED utility in either command or insert mode. ED displays the * prompt on the screen when ED is in command mode. When the * appears, you can enter the single letter command that reads text from the buffer, moves the CP, or changes the ED mode of operation. When in command mode, you can use the line-editing characters CTRL-C, CTRL-E, CTRL-H, CTRL-U, CTRL-X, and DEL to edit your input. In insert mode, however, you use only CTRL-H, CTRL-U, CTRL-X, and DEL.

Table 5-6. ED Command Summary

Command	*Action*
nA	Append n lines from original file to memory buffer.
0A	Append file until buffer is one half full.
#A	Append file until buffer is full (or end of file).
B, -B	Move CP to beginning (B) or bottom (-B) of buffer.

nC, -nC	Move CP n characters forward (C) or back (-C) through buffer.
nD, -nD	Delete n characters before (-D) or from (D) the CP.
E	Save new file and return to CP/M Plus.
Fstring{|Z}	Find character string.
H	Save the new file, then reedit, using the new file as the original file.
I	Enter insert mode; use ↑Z or ESCape to exit insert mode.
Istring{|Z}	Insert string at CP.

Note: upper-case I forces all input to upper-case; while lower-case i allows upper- and lower-case.

Table 5-6 (continued)

Jsearch-str^Zins-str^Zdel-to-str{|Z}	Juxtapose strings.
nK, -nK	Delete (kill) n lines from the CP.
nL, -nL, 0L	Move CP n lines.
nMcommands	Execute commands n times.
n, -n	Move CP n lines and display that line.
n:	Move to line n.
:ncommand	Execute command through line n.
Nstring^{ b|Z}	Extended find string.

O	Return to original file.
nP, -nP	Move CP n lines forward and display n lines at console.
Q	Abandon new file, return to CP/M Plus.
R{|Z}	Read X$$$$$$.LIB file into buffer.
Rfilespec{|Z}	Read filespec into buffer.
Sdelete string^Zinsert string{|Z}	Substitute string.
nT, -nT, 0T	Type n lines.
U, -U	Upper-case translation.

Table 5-6 (continued)

V, -V, 0V	Line numbering on/off, display free buffer space.
nW	Write n lines to updated file.
nX{|Z}	Write or append n lines to X$$$$$$.LIB.
nXfilespec{|Z}	Write n lines to filespec or append if previous x command applied to the same file.
0X{|Z}	Delete file X$$$$$$.LIB.
0Xfilespec{|Z}	Delete filespec.
nZ	Wait n seconds.

Section 6 gives a detailed description of the overall operation of the ED utility and the use of each command.

If you do not include a command tail in the ED command, it prompts you for the input filespec and the output filespec as follows:

```
Enter Input File:
```

After you enter the input filespec, ED prompts again:

```
Enter Output File:
```

Enter a filename or drive if you want the output file or its location to be different from that of the input file. Press RETURN if you want the output file to replace the input file. In this case, the input file is renamed to type BAK.

If the second file specification contains only the drive specifier, the second filename and filetype become the same as the first filename and filetype.

If the file given by the first file specification is not present, ED creates the file and writes the message:

```
NEW FILE
```

If the file given by the first filespec is already present, you must issue the A command to read portions of the file to the buffer. If the size of the file does not exceed the size of the buffer, the command

```
#a
```

reads the entire file to the buffer.

The i (Insert) command places ED in insert mode. In this mode, any characters you type are stored in sequence in the buffer starting at the current CP.

Any single letter commands typed in insert mode are not interpreted as commands, but are simply stored in the buffer. To return from insert mode to command mode, press CTRL-Z or the ESC key. Note that you can always substitute the ESC key for CTRL-Z in ED.

The single letter commands are usually typed in lower-case. The commands that must be followed by a character sequence end with CTRL-Z if they are to be followed by another command letter.
Any single letter command typed in upper-case tells ED to internally translate to upper-case all characters up to the CTRL-Z that ends the command.

When enabled, line numbers that appear on the left of the screen take the form:

```
nnnnn:
```

where nnnnn is a number in the range 1 through 65535. Line numbers are displayed for your reference and are not contained in either the buffer or the character file. The screen line starts with

```
:*
```

when the CP is at the beginning or end of the buffer.

Examples:

```
A>ED MYPROG.PAS
```

If not already present, this command line creates the file MYPROG.PAS on drive A. The command prompt

```
:*
```

appears on the screen. This tells you that the CP is at the beginning of the buffer. If the file is already present, issue the command

```
:**a
```

to fill the buffer. Then type the command

```
:*0p
```

to fill the screen with the first n lines of the buffer, where n is the current default page size (See the DEVICE command to set the page size).

Type the command

```
:*e
```

to stop the ED utility when you are finished changing the character file. The ED utility leaves the original file unchanged as MYPROG.BAK and the altered file as MYPROG.PAS.

```
A>ED MYPROG.PAS B:NEWPROG.PAS
```

The original file is MYPROG.PAS on the default drive A. The original file remains unchanged when the ED utility finishes, with the altered file stored as NEWPROG.PAS on drive B.

```
A>B:ED MYPROG.PAS B:
```

The ED.COM file must be on drive B. The original file is MYPROG.PAS located on drive A. It remains unchanged, with the altered program stored on drive B as MYPROG.PAS.

The ERASE Command

Syntax: ERASE {filespec} {[CONFIRM]}

Explanation: The ERASE command removes one or more files from a disk's directory in the current user number. Wildcard characters are accepted in the filespec. Directory and data space are automatically reclaimed for later use by another file. The ERASE command can be abbreviated to ERA.

Use the ERASE command with care because all files in the current user number that satisfy the file specification are removed from the disk directory.

Command lines that take the form

ERASE {d:}wildcard-filespec

require your confirmation because they erase an entire group of files, not just one file. The system prompts with the following message:

```
ERASE {d:}wildcard-filespec (Y/N)?
```

Respond with y if you want to remove all matching files, and n if you want to avoid erasing any files.

If no files match the file specification, you see the following message:

```
No File
```

The CONFIRM option informs the system to prompt for verification before erasing each file that matches the filespec. You can abbreviate CONFIRM to C.

If you use the CONFIRM option with wildcard-filespec, then ERASE prompts for confirmation for each file. You can selectively erase the files you want by responding Y to the confirm message, or keep the files by responding N to the confirm message.

Examples:

```
A>ERASE X.PAS
```

This command removes the file X.PAS from the disk in drive A.

```
A>ERA *.PRN
```

The system asks to confirm:

```
ERASE *.PRN (Y/N)?Y
```

All files with the filetype PRN are removed from the disk in drive A.

```
B>ERA A:MY*.* [CONFIRM]
```

Each file on drive A with a filename that begins with MY is displayed with a question mark for confirmation. Type Y to erase the file displayed, N to keep the file.

```
A>ERA B:*.*
ERASE B:*.* (Y/N)?Y
```

All files on drive B are removed from the disk.

The GENCOM Command

Syntax: GENCOM {COM-filespec} {RSX-filespec} ... {[LOADER|NULL|SCB=(offset,value)]}

Explanation: The GENCOM command is a transient utility that creates a special COM file with attached RSX files. RSX files are used as Resident System Extensions and are discussed in detail in part 2 of this manual. GENCOM places a special header at the beginning of the output program file to indicate to the system that RSX loading is required. It can also set a flag to keep the program loader active.

The GENCOM command can also restore a file already processed by GENCOM to the original COM file without the header and RSXs. GENCOM has three options that help you attach RSX files:

- The LOADER option sets a flag to keep the program loader active, (For complete details on the LOADER option, read about CP/M function 59 in Part 2 of this manual. This option is used only if no RSX files are attached to the COM file.

- The NULL option indicates that only RSX files are specified. GENCOM creates a dummy COM file for the RSX files. The output COM filename is taken from the filename of the first RSX-filespec.
- The SCB=(offset,value) option sets the System Control Block from the program by using the hex values specified by (offset,value). For complete details on the SCB option read about CP/M function 49 in Part 2 of this manual.

Attach RSX Files to a COM File

Syntax: GENCOM COM-filespec RSX-filespec ... {[LOADER|SCB=(offset,value)]}

Explanation: The preceding form of the GENCOM command creates a COM file with a header and attached RSXs. A maximum of 15 RSXs can be attached. GENCOM expects the first filespec to be a COM file and the following filespecs to be RSX files. Note that the original COM file is replaced by the newly-created COM file.

Example:

```
A>GENCOM MYPROG PROG1 PROG2
```

The preceding command generates a new COM file MYPROG.COM with attached RSXs PROG1 and PROG2.

Generate a COM File Using Only RSX Files

Syntax: GENCOM RSX-filespec {RSX-filespec} ... [NULL {SCB=(offset,value)}]

Explanation: The preceding form of the GENCOM command attaches the RSX files to a dummy COM file. GENCOM creates a COM file with the filename of the first RSX-filespec in the command tail. This format allows the system to load RSXs directly.

Example:

```
A>GENCOM PROG1 PROG2 [NULL]
```

The preceding command creates a COM file PROG1.COM with Resident System Extensions PROG1.RSX and PROG2.RSX.

Restore a File with Attached RSXs to Original COM File

Syntax: GENCOM filename

Explanation: The preceding form of the GENCOM file takes a file that has already been processed by GENCOM and restores it to its original COM file format. This form of the command assumes a filetype of COM.

Example:

```
A>GENCOM MYPROG
```

In the preceding command, GENCOM takes MYPROG.COM, strips off the header and deletes all attached RSXs to restore it to its original COM format.

Update (Add or Replace) RSX Files

Syntax: GENCOM COM-filespec RSX-filespec ... {[LOADER | SCB=(offset,value)]}

Explanation: The preceding form of the GENCOM command adds and/or replaces RSX files to a file already processed by GENCOM.

GENCOM inspects the list of RSX files. If they are new, they are added to the file already processed by GENCOM. If they already exist, then GENCOM replaces the existing RSXs with the new RSX files.

Example:

```
A>GENCOM MYPROG PROG1 PROG2
```

In the preceding example, GENCOM looks at MYPROG.COM, which is already processed by GENCOM, to see if PROG1.RSX and PROG2.RSX are already attached RSX files in the module. If either one is already attached, GENCOM replaces it with the new RSX module. Otherwise, GENCOM appends the specified RSX files to the COM file.

Attach a Header Record

Syntax: GENCOM filename [SCB=(offset,value),... | LOADER]

Explanation: The preceding syntax line attaches a GENCOM header record, with the SCB or loader flag set, to a file of type COM that contains no RSXs. This form of the command does not attach RSXs to a file.

Examples:

```
A>GENCOM FILETWO [loader]
```

The preceding command attaches a 256-byte header record to the file FILETWO.COM and sets the loader flag in the header record.

```
A>GENCOM FILEFOUR [scb=(1,1)]
```

The preceding command causes the program loader to set byte 1 of the System Control Block to 1 when it loads FILEFOUR.COM.

For more information, see functions 49, Set/Get System Control Block, and 59, Load Overlay or Resident System Extensions, in Part 2 of this manual.

The GET Command

Syntax:

```
GET {CONSOLE INPUT FROM} FILE filespec
                        {[{ECHO|NO ECHO}|SYSTEM]}
GET {CONSOLE INPUT FROM} CONSOLE
```

Explanation: The GET command is a transient utility that directs CP/M 3 to take console input from a file. The file can contain CP/M 3 system commands and/or input for a user program. If you use the SYSTEM option, GET immediately takes the next system command from the file.

Console input is taken from a file until the program terminates. If the file is exhausted before program input is terminated, the program looks for subsequent input from the console. If the program terminates before exhausting all its input, the system reverts back to the console for console input.

When the SYSTEM option is used, the system immediately goes to the file specified for console input. If you omit the SYSTEM option, you can enter one system command to initiate a user program whose console input is taken from the file specified in the GET command. The system reverts to the console for input when it reaches the end of the GET file input. The system also reverts to the console for console input if a GET CONSOLE INPUT FROM CONSOLE command is included in the input file.

Get Console Input from a File

Syntax:

```
GET {CONSOLE INPUT FROM} FILE filespec
{[options]}
```

Explanation: The preceding form of the GET command tells the system to get subsequent console input from a file. Table 5-7 lists the

GET options that you use in the following format:

[{ECHO | NO ECHO} | SYSTEM]

Table 5-7. GET Options

Option	*Meaning*
ECHO	specifies that the input is echoed to the console. This is the default option.
NO ECHO	specifies that the file input is not to be echoed to the console. The program output and the system prompts are not affected by this option and are still echoed to the console.
SYSTEM	specifies that all system input is to be taken from the disk file specified in the command line. GET takes system and program input from the file until the file is exhausted or until GET reads a GET console command from the file.

Examples:

```
A>GET FILE XINPUT
A>MYPROG
```

The preceding sequence of commands tells the system to activate the GET utility. However, because SYSTEM is not specified, the system reads the next input line from the console and executes MYPROG. If MYPROG program requires console input, it is taken from the file XINPUT. When MYPROG terminates, the system reverts to the console for console input.

```
A>GET FILE XIN2 [SYSTEM]
```

The preceding command immediately directs the system to get subsequent console input from file XIN2 because it includes the SYSTEM option. The system reverts to the console for console input when it reaches the end of file in XIN2. Or, XIN2 can redirect the system back to the console if it contains a GET CONSOLE command.

Terminate Console Input from a File

Syntax: GET {CONSOLE INPUT FROM} CONSOLE

Explanation: The preceding form of the GET command tells the system to get console input from the console.

Example:

```
A>GET CONSOLE
```

The preceding GET command tells the system to get console input from the console. You can use this command in a file (previously specified in a GET FILE command) which is already being read by the system for console input. It is used to redirect the console input to the console before the end of the file is reached.

The HELP Command

Syntax: HELP {topic}{subtopic1 subtopic2...subtopic8}{[NO PAGE|LIST]}
HELP [EXTRACT]
HELP [CREATE]

Explanation: The HELP command is a transient utility that provides summarized information for all of the CP/M Plus commands described in this manual. In the distributed CP/M Plus system, HELP presents general information on a command as a topic and detailed information on a command as a subtopic. HELP with no command tail displays a list of all the available topics. HELP with a topic in the command tail displays information about that topic, followed by any available subtopics. HELP with a topic and a subtopic displays information about the specific subtopic.

After HELP displays the information for your specified topic, it displays the special prompt HELP> on your screen. Subtopics can be accessed by preceding the subtopic with a period. The period causes the subtopic search to begin at the last known level. You can continue to specify topics for additional information, or simply press the RETURN key to return to the CP/M Plus system prompt.

You can abbreviate the names of topics and subtopics. Usually one or two letters is enough to specifically identify the topics.

Display Information

Syntax: HELP topic {subtopic1...subtopic8}{[NOPAGE|LIST]}
HELP.Subtopic

Explanation: The preceding forms of the HELP command display the information for the specified topic and subtopics. Use the following two options with this form of the HELP command:

- The NOPAGE option disables the default paged display of every n lines, where n is the number of lines per page as set by the system or as set by the user. To stop the display, press CTRL-S. To resume the display, press CTRL-Q. You can abbreviate NOPAGE to N. (See the DEVICE command for more information about setting the number of lines per page.)

- The LIST option is the same as NOPAGE, except that it eliminates extra lines between headings. Use this option with CTRL-P to list the help information on the printer.

Examples:

```
A>HELP
```

The preceding command displays a list of topics for which help is available.

```
A>HELP DATE
```

This command displays general information about the DATE command. It also displays any available subtopics.

```
A>HELP DIR OPTIONS [N]
```

The preceding command includes the subtopic options. In response, HELP displays information about options associated with the DIR command. The display is not in paged mode.

```
A>HELP ED
```

The preceding command displays general information about the ED utility.

```
A>HELP ED COMMANDS
```

This form of HELP displays information about commands internal to ED. The preceding example can also be entered as

```
A>HELP ED
HELP>.COMMANDS
```

Add Your Own Descriptions to the HELP.HLP File

Syntax: HELP [EXTRACT]
HELP [CREATE]

Explanation: CP/M Plus is distributed with two related HELP files: HELP.COM and HELP.HLP. The HELP.COM file is the command file that processes the text of the HELP.HLP file and displays it on the screen. The HELP.HLP file is a text file to which you can add customized information, but you cannot directly edit the HELP.HLP file. You must use the HELP.COM file to convert HELP.HLP to a file named

HELP.DAT before you can edit or add your own text.

This form of the HELP command has the following options:

- The EXTRACT option accesses the file HELP.HLP on the default drive and creates a file called HELP.DAT on the default drive. You can now invoke a word processing program to edit or add your own text to the HELP.DAT file. EXTRACT can be abbreviated to E.

- The CREATE option accesses your edited HELP.DAT file on the default drive and builds a revised HELP.HLP file on the default drive. CREATE can be abbreviated to C.

You must add topics and subtopics to the HELP.DAT file in a specific format. A topic heading in the HELP.DAT file takes the form:

```
///nTopicname<cr>
```

The three backslashes are the topic delimiters and must begin in column one. In the preceding format statement, n is a number in the range from 1 through 9 that signifies the level of the topic. A main topic always has a level number of 1. The first subtopic has a level number of 2. The next level of subtopic has a level number of 3, and so forth, up to a maximum of nine levels. Topicname is the name of your topic, and allows a maximum of twelve characters. The entire line is terminated with a carriage return.

Use the following guidelines to edit and insert text into the HELP.DAT file.

- Topics should be placed in alphabetical order.

- Subtopics should be placed alphabetically within their respective supertopic.

- Levels must be indicated by a number 1-9.

Some examples of topic and subtopic lines in the HELP.HLP file follow:

```
///1NEW UTILITY<cr>

///2COMMANDS<cr>

///3PARAMETERS<cr>

///2EXAMPLES<cr>
```

The first example illustrates the format of a main topic line. The second example shows how to number the first subtopic of that main topic. The third example shows how the next level subtopic under level 2 should be numbered. The fourth example shows how to return to the lower level subtopic. Any topic name with a level number of 1 is a main topic. Any topic name with a level number of 2 is a subtopic within its main topic.

When you are executing the HELP.COM file, you need only enter enough letters of the topic to unambiguously identify the topic name. When referencing a subtopic, you must type the topic name AND the subtopic, otherwise the HELP program cannot determine which main topic you are referencing. You can also enter a topic and subtopic following the program's internal prompt, HELP>, as follows:

```
HELP>ED COMMANDS
```

This form of HELP displays information about commands internal to the editing program, ED.

The HEXCOM Command

Syntax: HEXCOM filename

Explanation: The HEXCOM command is a transient utility that generates a command file (filetype COM) from a HEX input file. It names the output file with the same filename as the input file but with filetype COM. HEXCOM always looks for a file with filetype HEX.

Example:

```
A>HEXCOM B:PROGRAM
```

In the preceding command, HEXCOM generates a command file PROGRAM.COM from the input hex file PROGRAM.HEX.

The INITDIR Command

Syntax: INITDIR d:

Explanation: The INITDIR command can initialize a disk directory to allow date and time stamping of files on that disk or remove date and time stamps.

You must use INITDIR to initialize the directory for any disk on which you plan to record date and time stamps for your files. If the disk is blank, INITDIR initializes the directory to record date and time stamps. If files already exist on the disk, INITDIR checks the space available for date and time stamps in the directory. If there is not enough room for date and time stamps, INITDIR does not initialize the directory and returns an error message.

After you initialize the directory for date and time stamps, you must use the SET command to specify time stamp options on the disk.

Note that Locoscript 1.20 does not update access time. Earlier versions cannot use disks with extended directories.

Examples:

```
A>INITDIR B:
```

The system prompts to confirm:

```
INITDIR WILL ACTIVATE TIME STAMPS FOR SPECIFIED
DRIVE.
Do you really want to re-format the directory: B (Y/N)?
```

If the directory has previously been initialized for date and time stamps, INITDIR displays the message:

```
Directory already re-formatted
Do you wish to recover date/time directory space (Y/N)?
```

Enter Y to reinitialize the directory to eliminate date and time stamps. If you enter N, date and time stamping remains active on your disk and INITDIR displays the following message:

```
Do you want the existing date/time stamps cleared (Y/N)?
```

Enter Y to clear the existing stamps. Enter N to keep the existing date and time stamps.

The LANGUAGE Command

Syntax: LANGUAGE number

Explanation: The LANGUAGE command enables you to select the character set you require for working in a particular language. There are 8 language variants available:

0 American English
1 French
2 German
3 English
4 Danish
5 Swedish

6 Italian
7 Spanish

The default is 0 for Great Britain.

Note that the LANGUAGE command, in order to achieve certain characters, may change the ASCII character to HEX mappings.

Full details of the character sets, and their keyboard positions are given in the User Guide that was supplied with your PCW8256 or CPC6128.

Example:

```
A>LANGUAGE 6
```

This command sets the character set for Italian.

The LIB Command

Syntax:

```
LIB filespec{[I|M|P|D]}
LIB filespec{[I|M|P]}=filespec{modifier}
                      {,filespec{modifier} ... }
```

Explanation: A library file contains a collection of object modules. Use the LIB utility to create libraries, and to append, replace, select, or delete modules from an existing library. You can also use LIB to obtain information about the contents of library files.

LIB creates and maintains library files that contain object modules in MicroSoft® REL format. These modules are produced by Digital Research's relocatable macro-assembler program, RMAC, or any other language translator that produces modules in MicroSoft REL format.

LINK-80™ links the object modules contained in a library to other object files. LINK-80 automatically selects from the library only those modules needed by the program being linked, and then forms an executable file with a filetype of COM.

The library file has the filetype REL or IRL depending on the option you choose. Modules in a REL library file must not contain backward references to modules that occur earlier in the library, because LINK-80 currently makes only one pass through a library.

Table 5-8. LIB Options

Option	*Meaning*
I	The INDEX option creates an indexed library file of type IRL. LINK-80 searches faster on indexed libraries than on nonindexed libraries.
M	The MODULE option displays module names.
P	The PUBLICS option displays module names and the public variables for the new library file.
D	The DUMP option displays the contents of object modules in ASCII form.

Use modifiers in the command line to instruct LIB to delete, replace, or select modules in a library file. Angle brackets enclose the modules to be deleted or replaced. Parentheses enclose the modules to be selected.

Unless otherwise specified, LIB assumes a filetype of REL for all source filenames. When you follow a filename by a group of module names enclosed in parentheses, these modules are included in the new library file. If modules are not specified, LIB includes all modules from the source file in the new library file.

Table 5-9. LIB Modifiers

Modifier	*Meaning*
Delete	<module=>
Replace	<module=filename.REL> If module name and filename are the same this shorthand can be used: <filename>
Select	(modFIRST-modLAST,mod1,mod2,..., modN)

Examples:

```
A>LIB TEST4[P]
A>LIB TEST5[P]=FILE1,FILE2
```

The first example displays all modules and publics in TEST4.REL. The second example creates TEST5.REL from FILE1.REL and FILE2.REL, and displays all modules and publics in TEST5.REL.

```
A>LIB TEST=TEST1(MOD1,MOD4),TEST2(C1-C4,C6)
```

In the preceding example LIB creates a library file TEST.REL from modules in two source files. TEST1.REL contributes MOD1 and MOD4. LIB extracts modules C1, C4, all the modules located between them, and module C6 from TEST2.REL.

```
A>LIB FILE2=FILE3<MODA=>
```

In this example, LIB creates FILE2.REL from FILE3.REL, omitting MODA which is a module in FILE3.REL.

```
A>LIB FILE6=FILE5<MODA=FILEB.REL>
A>LIB FILE6=FILE5<THISNAME>
```

In the first example, MODA is in the existing FILE5.REL. When LIB creates FILE6.REL from FILE5.REL, FILEB.REL replaces MODA.

In the second example, module THISNAME is in FILE5.REL. When LIB creates FILE6.REL from FILE5.REL the file THISNAME.REL replaces the similarly named module THISNAME.

```
A>LIB
FILE1=B:FILE2(PLOTS,FIND,SEARCH-DISPLAY)
```

In this example LIB creates FILE1.IRL on drive A from the selected modules PLOTS, FIND, and modules SEARCH through the module DISPLAY, in FILE2.REL on drive B.

The LINK Command

Syntax: LINK d:{filespec,{[o]}=}filespec{[o]}{,...}

Explanation: The LINK command combines relocatable object modules such as those produced by RMAC and PL/I-80™ into a .COM file ready for execution. Relocatable files can contain external references and publics. Relocatable files can reference modules in library files. LINK searches the library files and includes the referenced modules in the output file. The LINK command is the LINK-80 utility and are synonymous in this discussion. See the Programmer's Utilities Guide for the CP/M Family of Operating Systems for a complete description of LINK-80.

You can use LINK option switches to control the execution parameters of LINK-80. Link options follow the file specifications and are enclosed within square brackets. Multiple switches are separated by commas.

Table 5-10. LINK Options

Option	*Meaning*
A	Additional memory; reduces buffer space and writes temporary data to disk.
B	BIOS link in banked CP/M Plus system. Aligns data segment on page boundary; puts length of code segment in header; defaults to SPR filetype.
Dhhhh	Data origin; sets memory origin for common and data area.
Gn	Go; set start address to label n.
Lhhhh	Load; change default load address of module to hhhh. Default 0100H.
Mhhhh	Memory size; define free memory requirements for MP/M™ modules.
NL	No listing of symbol table at console.
NR	No symbol table file.
OC	Output COM command file. Default.
OP	Output PRL page relocatable file for execution under MP/M in relocatable segment.
OR	Output RSP Resident System Process file for execution under MP/M.
Phhhh	Program origin; changes default program origin address to hhhh. Default is 0100H.
Q	Lists symbols with leading question mark.

S	Search preceding file as a library.
$Cd	Destination of console messages, d, can be X for console, Y for printer, or Z for zero output. Default is X.
$Id	Source of intermediate files; d is disk drive A-P. Default is current drive.
$Ld	Source of library files; d is disk drive A-P. Default is current drive.
$Od	Destination of object file; d can be Z, or disk drive A-P. Default is to same drive as first file in the LINK-80 command.
$Sd	Destination of symbol file; d can be Y, Z, or disk drive A-P. Default is to same drive as first file in LINK-80 command.

Examples:

```
A>LINK b:MYFILE[NR]
```

LINK-80 on drive-A uses as input MYFILE.REL on drive B and produces the executable machine code file MYFILE.COM on drive B. The [NR] option specifies no symbol table file.

```
A>LINK m1,m2,m3
```

LINK-80 combines the separately compiled files m1, m2, and m3, resolves their external references, and produces the executable machine code file m1.COM.

```
A>LINK m=m1,m2,m3
```

LINK-80 combines the separately compiled files m1, m2, and m3 and produces the executable machine code file m.COM.

```
A>LINK MYFILE,FILE5[s]
```

The [s] option tells LINK-80 to search FILE5 as a library. LINK-80 combines MYFILE.REL with the referenced subroutines contained in FILE5.REL on the default drive A and produces MYFILE.COM on drive A.

The MAC Command

Syntax: MAC filename {$options}

Explanation: MAC, the CP/M Macro Assembler, is a transient utility that reads assembly language statements from a disk file of filetype ASM. MAC assembles the statements and produces three output files with the input filename and output filetypes of HEX, PRN, and SYM.

Filename.HEX contains Intel® hexadecimal format object code. You can debug the HEX file with a debugger, or use HEXCOM to create a COM file and execute it.

Filename.PRN contains an annotated source listing that can be printed or examined at the console. The PRN file includes a 16-column wide listing at the left side of the page that shows the values of literals, machine code addresses, and generated machine code. An equal sign denotes literal addresses to eliminate confusion with machine code addresses.

Filename.SYM contains a sorted list of symbols defined in the program.

Before invoking MAC, you must prepare a source program file with the filetype ASM containing assembly language statements.

You can direct the input and output of MAC using the options listed in the following table. Use a letter with the option to indicate the source and destination drives, console, printer, or zero output. Valid drive names are A through O. X directs output to the console. P directs output to the printer. Z specifies that output files will not be created.

Table 5-11. Input/Output Options

Option	*Meaning*
A	source drive for ASM file (A-O)
H	destination drive for HEX file (A-O, Z)
L	source drive for macro library LIB files called by the MACLIB statement.
P	destination drive for PRN file (A-O, X, P, Z)
S	destination drive for SYM file (A-O, X, P, Z)

Table 5-12. Output File Modifiers

Modifier	*Meaning*
+L	lists input lines read from macro library LIB files
-L	suppresses listing (default)
+M	lists all macro lines as they are processed during assembly
−M	suppresses all macro lines as they are read during assembly
*M	lists only hex generated by macro expansions
+Q	lists all LOCAL symbols in the symbol list
-Q	suppresses all LOCAL symbols in the symbol list (default)

Table 5-12 (continued)

+S	appends symbol file to print file
-S	suppresses creation of symbol file
+1	produces a pass 1 listing for macro debugging in PRN file
-1	suppresses listing on pass 1 (default)

Examples:

```
A>MAC SAMPLE
```

In the preceding example MAC is invoked from drive A and operates on the file SAMPLE.ASM also on drive A.

```
A>MAC SAMPLE $PB AA HB SX
```

In this example, an assembly option parameter list follows the MAC command and the source filename. The parameters direct the PRN file to drive B, obtain the ASM file from drive A, direct the HEX file to drive B, and send the SYM file to the console. You can use blanks between option parameters.

The PALETTE Command

Syntax: PALETTE number number number

Explanation: The first number specifies the colour of ink 0, the second number specifies ink 1 and so on until either all inks have been specified or the list of colours is exausted.

Each colour is given as a number in the range 0..63. The colour number represents three 2 bit numbers each corresponding to the intensity of one of the primary colours, bits 4, 5 for green, bits 2,3 for red and bits 0,1 for blue.

CPC6128

There are 15 inks and three levels of colour intensity; these are mapped onto the required four levels of intensity as follows:

colour	: 0 1 2 3
CPC6128	: 0 1 1 2

intensity 3 is interpreted as intensity 2,

PCW8256

The PCW8256 has a monochromatic screen. There are two inks 0 and 1. If the colour of ink 0 is greater than ink 1 then the screen is displayed in inverse video, black characters on a white background, otherwise white characters on a black background.

Any colour number greater than 63 is masked with 63.

If more colours than inks are given the remainder are ignored. Full details of the PALETTE Command are given in the User Guide for your machine.

Examples: *PCW8256*

PALETTE 1 0 sets reverse video (black characters on a light background)

CPC6128

PALETTE 63 1 sets a bright white background with text in blue

The PAPER Command

Syntax: PAPER parameter parameter ...

Explanation: Note that this command is used only with the PCW8256 and sets printer parameters for the PCW8256 printer or an Epson FX-80.

To use the printer effectively it is necessary to tell it the length of paper in use, whether it is single sheet or continuous, and so on. The PAPER utility allows the operator to set these parameters.

PAPER is a program which takes a number of parameters from the command line which invokes it. Once the parameters have been checked for validity the program sends suitable escape sequences to the printer (via the CP/M LST: calls). Note that this means that firstly it may be used with almost any Epson compatible printer and secondly it requires the printer to be ready to accept characters. After each escape sequence is sent to the printer PAPER reports on the console what it has sent.

The parameters for PAPER are:

Form Length <number>

The <number> must be in the range 6..99, and sets the form length in lines. If the Line Pitch is not set explicitly in this use of PAPER, then it is set to "standard line pitch" (six lines per inch). If a Gap Length is not set explicitly in this use of PAPER then the gap length is set to zero.

Gap Length <number>

The <number> must be in the range 0..99, and sets the gap length in lines. If the gap length specified is not zero and the Line Pitch is not set explicitly in this use of PAPER, then it is set to "standard line pitch" (six lines per inch).

Line Pitch <number>

The <number> may be 6 or 8, setting 6 or 8 lines per inch.

Single Sheet

Sets single sheet stationery. If Paper Out Defeat is not set explicitly in this use of PAPER then it is set On.

Continuous Stationery

Sets continuous stationry. If Paper Out Defeat is not set explicitly in this use of PAPER then it is set Off.

Paper Out Defeat On or Paper Out Defeat Off

Sets paper out defeat as required.

Defaults

Tells the printer to copy its current settings (including those given in this use of PAPER) to its memory of default settings.

The program in fact requires only the first letter of each of the keywords given above, except for On and Off which must be given in full. In all cases only the first keyword is required, the others are optional.

Three further parameters are accepted:

A4 or A5

These set : 6 lines per inch Form length 70 lines (A4) or 50 lines (A5)
Gap length 3 lines
Single Sheet Paper out defeat On

<number>

This must be a number in the range 1..17 and is provided to set up for continuous stationery, with a form length as given measured in inches.

The following are set :	6 lines per inch
	Form length <number> inches
	Gap length 0
	Continuous stationery Paper out defeat Off

Details of using the PAPER command are given in your PCW8256 User Guide.

The PATCH Command

Syntax: PATCH filename {typ} {n}

Explanation: The PATCH command displays or installs patch number n to the CP/M Plus system or CP/M Plus command files.

Only CP/M Plus system files of filetype COM, PRL, or SPR can be patched with the PATCH command. If the typ option is not specified, the PATCH utility looks for a file with a filetype of COM.

The patch number n must be between 1 and 32 inclusive.

Examples:

```
A>PATCH SHOW 2
```

The preceding command patches the system SHOW.COM file with patch number 2. The system displays the following question:

```
Do you want to indicate that Patch #2 has been installed for
SHOW.COM?Y
```

If the patch is successful, the system displays the message:

```
Patch Installed
```

If the patch is not successful, the system displays the following message:

```
Patch not Installed
```

One of the following error messages might be displayed:

- ERROR: Patch requires CP/M Plus.
- ERROR: Invalid filetype typ.
- ERROR: Serial Number mismatch.
- ERROR: Invalid patch number n.

The PIP Command

Syntax: PIP dest-filespec|d:{[Gn]}=src-filespec{[o]}{,...} | d: {[o]}

Explanation: PIP is a transient utility that copies one or more files from one disk and, or user number to another. PIP can rename a file after copying it. PIP can combine two or more files into one file. PIP can also copy a character file from disk to the printer or other auxiliary logical output device. PIP can create a file on disk from input from the console or other logical input device. PIP can transfer data from a logical input device to a logical output device, thus the name Peripheral Interchange Program.

PIP copies file attributes with the file. This includes Read-Write or Read-Only and SYS or DIR file attributes and the user-definable attributes F1 through F4. If a file is password-protected, you must enter the password in the command line following the filename and/or filetype to which it belongs. If the password fails, the file is skipped and the failure noted.

When you specify a destination file with a password, PIP assigns that password to the destination file and automatically sets the password protection mode to READ.

When you specify a destination file with no password, PIP does not assign a password to the destination file. When you specify only a destination drive, PIP assigns the same password and password protection mode to the destination file as specified in the source file. When you specify a destination file with a password, PIP automatically sets the password protection mode to READ. This means that you need a password to read the file. (See the SET command.)

Single File Copy

Syntax: PIP d:{[Gn]} = src-filespec{[options]}

PIP dest-filespec{[Gn]} = d:{[options]}

PIP dest-filespec{[Gn]} = src-filespec{[o]}

Explanation: The first form shows the simplest way to copy a file. PIP looks for the file named by src-filespec on the default or optionally specified drive. PIP copies the file to the drive specified by d: and gives it the name specified by src-filespec. If you want, you can use the [Gn] option to place your destination file (dest-filespec) in the user number specified by n. The only option recognized for the destination file is [Gn]. Several options can be combined together for the source file specification (src filespec). See Table 5-13, PIP options.

The second form is a variation of the first. PIP looks for the file named by dest-filespec on the drive specified by d:, copies it to the default or optionally specified drive, and gives it the name specified by dest-filespec.

The third form shows how to rename the file after you copy it. You can copy it to the same drive and user number, or to a different drive and/or user number. Rules for options are the same. PIP looks for the file specified by src-filespec, copies it to the location specified in dest-filespec, and gives it the name indicated by dest-filespec.

Remember that PIP always goes to and gets from the current default user number unless you specify otherwise with the [Gn] option.

Before you start PIP, be sure that you have enough free space in kilobytes on your destination disk to hold the entire file or files that you are copying. Even if you are replacing an old copy on the destination disk with a new copy, PIP still needs enough room for the new copy before it deletes the old copy. Use the DIR command to determine filesize and the SHOW command to determine disk space. If there is not enough space, you can delete the old copy first by using the ERASE command.

Data is first copied to a temporary file to ensure that the entire data file can be constructed in the space available on the disk. PIP gives the temporary file the filename specified for the destination, with the filetype $$$. If the copy operation is successful, PIP changes the temporary filetype $$$ to the filetype specified in the destination.

If the copy operation succeeds and a file with the same name as the destination file already exists, the old file with the same name is erased before renaming the temporary file.
File attributes (DIR, SYS, RO, RW) are transferred with the files.

If the existing destination file is set to Read-Only (RO), PIP asks you if you want to delete it. Answer Y or N. Use the [W] option to write over Read-Only files.

You can include PIP options following each source name. There is one valid option ([Gn] — go to user number n) for the destination file specification. Options are enclosed in square brackets. Several options can be included for the source files. They can be packed together or separated by spaces. Options can verify that a file was copied correctly, allow PIP to read a file with the system (SYS) attribute, cause PIP to write over Read-Only files, cause PIP to put a file into or copy it from a specified user number, transfer from lower- to upper-case, and much more.

Examples:

```
A>PIP B:=A:oldfile.dat
A>PIP B:oldfile.dat = A:
```

Both forms of this command cause PIP to read the file oldfile.dat from drive A and put an exact copy of it onto drive B. This is called the short form of PIP, because the source or destination names only a drive and does not include a filename. When using this form you cannot copy a file from one drive and user number to the same drive and user number. You must put the destination file on a different drive or in a different user number. (See the section on PIP Options, and the USER Command.) The second short form produces exatly the same result as the first one. PIP looks for the file oldfile.dat on drive A, the drive specified as the source.

```
A>PIP B:newfile.dat=A:oldfile.dat
```

This command copies the file oldfile.dat from drive A to drive B and renames it to newfile.dat. The file remains as oldfile.dat on drive A. This is the long form of the PIP command, because it names a file on both sides of the command line.

```
A>PIP newfile.dat = oldfile.dat
```

Using this long form of PIP, you can copy a file from one drive and user number (usually user 0 because CP/M Plus automatically starts out in user 0 — the default user number) to the same drive and user number. This gives you two copies of the same file on one drive and user number, each with a different name.

```
A>PIP B:PROGRAM.BAK = A:PROGRAM.DAT[G1]
```

The preceding command copies the file PROGRAM.DAT from user 1 on drive A to the current selected user number on drive B and renames the filetype on drive B to BAK.

```
B>PIP program2.dat = A:program1.dat[E V G3 f0]
```

In this command, PIP copies the file named program 1.dat on drive A and echoes [E] the transfer to the console, verifies [V] that the two copies are exactly the same, and gets [G3] the file program1.dat from user 3 on drive A. Because there is no drive specified for the destination, PIP automatically copies the file to the default user number and drive, in this case user 0 and drive B.

Multiple File Copy

Syntax: PIP d:{[Gn]} = {d:} wildcard-filespec {[options]}

Explanation: When you use a wildcard in the source specification, PIP copies matching files one-by-one to the destination drive, retaining the original name of each file. PIP displays the message COPYING followed by each filename as the copy operation proceeds. PIP issues an error message and aborts the copy operation if the destination drive and user number are the same as those specified in the source.

Examples:

```
A>PIP B:=A:*.COM
```

This command causes PIP to copy all the files on drive A with the filetype COM to drive B.

```
A>PIP B:=A:*.*
```

This command causes PIP to copy all the files on drive A to drive B. You can use this command to make a back-up copy of your distribution disk. Note, however, that this command does not copy the CP/M Plus system from the system tracks. COPYSYS copies the system for you.

```
A>PIP B:=A:PROG????.*
```

The preceding command copies all files whose filenames begin with PROG from drive A to drive B.

```
A>PIP B:[G1]=A:*.BAS
```

This command causes PIP to copy all the files with a filetype of BAS on drive A in the default user number (user 0) to drive B in user number 1. Remember that the DIR, TYPE, ERASE, and other commands only access files in the same user number from which they were invoked. (See the USER Command.)

Combining Files

Syntax: PIP dest-filespec{[Gn]} = src-filespec{[o]}, src-filespec{[o]}{,...}

Explanation: This form of the PIP command lets you specify two or more files in the source. PIP copies the files specified in the source from left to right and combines them into one file with the name indicated by the destination file specification. This procedure is called file concatenation. You can use the [Gn] option after the destination file to place it in the user number specified by n. You can specify one or more options for each source file.

Some of the options force PIP to copy files character-by-character. In these cases, PIP looks for a CTRL-Z character to determine where the end of the file is. All of the PIP options force a character transfer except the following:

A, C, Gn, K, O, R, V, and W.

Copying data to or from logical devices also forces a character transfer.

You can terminate PIP operations by typing CTRL-C.

When concatenating files, PIP only searches the last record of a file for the CTRL-Z end-of-file character. However, if PIP is doing a character transfer, it stops when it encounters a CTRL-Z character.

Use the [O] option if you are concatenating machine code files. The [O] option causes PIP to ignore embedded CTRL-Z (end-of-file) characters, which indicate the end-of-file character in text files, but might be valid data in object code files.

Examples:

```
A>PIP NEWFILE=FILE1,FILE2,FILE3
```

The three files named FILE1, FILE2, and FILE3 are joined from left to right and copied to NEWFILE.$$$. NEWFILE.$$$ is renamed to NEWFILE upon successful completion of the copy operation. All source and destination files are on the disk in the default drive A.

```
A>PIP B:X.BAS = Y.BAS, B:Z.BAS
```

The file Y.BAS on drive A is joined with Z.BAS from drive B and placed in the temporary file X.$$$ on drive B. The file X.$$$ is renamed to X.BAS on drive B when PIP runs to successful completion.

Copy Files to and from Auxiliary Devices

Syntax:

```
PIP dest-filespec {[Gn]} = src-filespec {[o]}
      AUX:                 AUX: {[o]}
      CON:                 CON: {[o]}
      PRN:                 NUL:
      LST:                 EOF:
```

Explanation: This form is a special case of the PIP command line that lets you copy a file from a disk to a device, from a device to a disk or from one device to another. The files must contain printable characters. Each peripheral device is assigned to a logical device that identifies a source device that can transmit data or a destination device that can receive data. (See the DEVICE command.) A colon follows each logical device name so it cannot be confused with a filename. Enter CTRL-C to abort a copy operation that uses a logical device in the source or destination.

The logical device names are listed as follows:

- CON: Console input or output device. When used as a source, usually the keyboard; when used as a destination, usually the screen.
- AUX: Auxiliary Input or Output Device.
- LST: The destination device assigned to the list output device, usually the printer.

The following three device names have special meaning:

- NUL: A source device that produces 40 hexadecimal zeros.
- EOF: A source device that produces a single CTRL-Z, the CP/M Plus end-of-file mark.
- PRN: The printer device with tab expansion to every eighth column, line numbers, and page ejects every sixtieth line.

Examples:

```
B>PIP PRN:=CON:,MYDATA.DAT
```

Characters are first read from the console input device, generally the keyboard, and sent directly to your printer device. You type a CTRL-Z character to tell PIP that keyboard input is complete. At that time, PIP continues by reading character data from the file MYDATA.DAT on drive B. Because PRN: is the destination device, tabs are expanded, line numbers are added, and page ejects occur every sixty lines.

Note that when the CON: device is the source you must enter both the carriage return (RETURN) and line-feed (LF) keys for a new line.

```
A>PIP B:FUNFILE.SUE = CON:
```

Whatever you type at the console is written to the file FUNFILE.SUE on drive B. End the keyboard input by typing a CTRL-Z.

```
A>PIP LST:=CON:
```

Whatever you type at the console keyboard is written to the list device, generally the printer. Terminate input with a CTRL-Z.

```
A>PIP LST:=B:DRAFT.TXT[T8]
```

The file DRAFT.TXT on drive B is written to the printer device. Any tab characters are expanded to the nearest column that is a multiple of 8.

```
A>PIP PRN:=B:DRAFT.TXT
```

The preceding command causes PIP to write the file DRAFT.TXT to the list device. It automatically expands the tabs, adds line numbers, and ejects pages after sixty lines.

Multiple Command Mode

Syntax: PIP

Explanation: This form of the PIP command starts the PIP utility and lets you type multiple command lines while PIP remains in user memory.

PIP writes an asterisk on your screen when ready to accept input command lines.

You can type any valid command line described under previous PIP formats following the asterisk prompt.

Terminate PIP by pressing only the RETURN key following the asterisk prompt. The empty command line tells PIP to discontinue operation and return to the CP/M Plus system prompt.

Examples:

```
A>PIP
CP/M Plus PIP VERSION 3.0
*NEWFILE=FILE1,FILE2,FILE3
```

```
*APROG.COM=BPROG.COM
*A:=B:X.BAS
*B:=*.*
*<RETURN>
*A>
```

This command loads the **PIP** program. The PIP command input prompt, *, tells you that PIP is ready to accept commands. The effects of this sequence of commands are the same as in the previous examples, where the command line is included in the command tail. PIP is not loaded into memory for each command. To exit this PIP command mode, press RETURN or one of its equivalent control characters, CTRL-J or CTRL-M as shown.

Using Options with PIP

Explanation: With options you can process your source file in special ways. You can expand tab characters, translate from upper- to lower-case, extract portions of your text, verify that the copy is correct, and much more.

The PIP options are listed in Table 5-13 using n to represent a number and s to represent a sequence of characters terminated by a CTRL-Z. An option must immediately follow the file or device it affects. The option must be enclosed in square brackets []. For those options that require a numeric value, no blanks can occur between the letter and the value.

You can include the [Gn] option after a destination file specification. You can include a list of options after a source file or source device. An option list is a sequence of single letters and numeric values that are optionally separated by blanks and enclosed in square brackets [].

Table 5-13. PIP Options

Option	*Function*
A	Copy only the files that have been modified since the last copy. To back up only the files that have been modified since the last back-up, use PIP with the archive option, [A].
C	Prompt for confirmation before performing each copy operation. Use the [C] option when you want to copy only some files of a particular filetype.
Dn	Delete any characters past column n. This parameter follows a source file that contains lines too long to be handled by the destination device, for example, an 80-character printer or narrow console. The number n should be the maximum column width of the destination device.
E	Echo transfer at console. When this parameter follows a source name, PIP displays the source data at the console as the copy is taking place. The source must contain character data.
F	Filter form-feeds. When this parameter follows a source name, PIP removes all form-feeds embedded in the source data. To change form-feeds set for one page length in the source file to another page length in the destination file, use the F command to delete the old form-feeds and a P command to simultaneously add new form-feeds to the destination file.
Gn	Get source from or go to user number n. When this parameter follows a source name, PIP searches the directory of user number n for the source file. When it follows the destination name, PIP places the destination file in the

Table 5-13 (continued)

	user number specified by n. The number must be in the range 0 to 15.
H	Hex data transfer. PIP checks all data for proper Intel hexadecimal file format. The console displays error messages when errors occur.
I	Ignore :00 records in the transfer of Intel hexadecimal format file. The I option automatically sets the H option.
L	Translate upper-case alphabetics in the source file to lower-case in the destination file. This parameter follows the source device or filename.
N	Add line numbers to the destination file. When this parameter follows the source filename, PIP adds a line number to each line copied, starting with 1 and incrementing by one. A colon follows the line number. If N2 is specified, PIP adds leading zeros to the line number and inserts a tab after the number. If the T parameter is also set, PIP expands the tab.
O	Object file transfer for machine code (noncharacter and therefore nonprintable) files. PIP ignores any CTRL-Z end-of-file during concatenation and transfer. Use this option if you are combining object code files.
Pn	Set page length. n specifies the number of lines per page. When this parameter modifies a source file, PIP includes a page eject at the beginning of the destination file and at every n lines. If n = 1 or is not specified, PIP inserts page ejects every sixty lines. When you also

	specify the F option, PIP ignores form-feeds in the source data and inserts new form-feeds in the destination data at the page length specified by n.
Qs	Quit copying from the source device after the string s. When used with the S parameter, this parameter can extract a portion of a source file.The string argument must be terminated by CTRL-Z.
R	Read system (SYS) files. Usually, PIP ignores files marked with the system attribute in the disk directory. But when this parameter follows a source filename, PIP copies system files, including their attributes, to the destination.
Ss	Start copying from the source device at the string s. The string argument must be terminated by CTRL-Z. When used with the Q parameter, this parameter can extract a portion of a source file. Both start and quit strings are included in the destination file.
Tn	Expand tabs. When this parameter follows a source filename, PIP expands tab (CTRL-I) characters in the destination file. PIP replaces each CTRL-I with enough spaces to position the next character in a column divisible by n.
U	Translate lower-case alphabetic characters in the source file to upper-case in the destination file. This parameter follows the source device or filename.
V	Verify that data has been copied correctly. PIP compares the destination to the source data to ensure that the data has been written correctly. The destination must be a disk file.

Table 5-13 (continued)

W	Write over files with RO (Read-Only) attribute. Usually, if a PIP command tail includes an existing RO file as a destination, PIP sends a query to the console to make sure you want to write over the existing file. When this parameter follows a source name, PIP overwrites the RO file without a console exchange. If the command tail contains multiple source files, this parameter need follow only the last file in the list.
Z	Zero the parity bit. When this parameter follows a source name, PIP sets the parity bit of each data byte in the destination file to zero. The source must contain character data.

Examples:

```
A>PIP NEWPROG.BAS=CODE.BAS[L], DATA.BAS[U]
```

This command constructs the file NEWPROG.BAS on drive A by joining the two files CODE.BAS and DATA.BAS from drive A. During the copy operation, CODE.BAS is translated to lower-case, while DATA.BAS is translated to upper-case.

```
A>PIP CON:=WIDEFILE.BAS[D80]
```

This command writes the character file WIDEFILE.BAS from drive A to the console device, but deletes all characters following the 80th column position.

```
A>PIP B:=LETTER.TXT[E]
```

The file LETTER.TXT from drive A is copied to LETTER.TXT on drive B. The LETTER.TXT file is also written to the screen as the copy operation proceeds.

```
A>PIP LST:=B:LONGPAGE.TXT[FP65]
```

This command writes the file LONGPAGE.TXT from drive B to the printer device. As the file is written, form-feed characters are removed and reinserted at the beginning and every 65th line thereafter.

```
B>PIP LST:=PROGRAM.BAS[NT8U]
```

This command writes the file PROGRAM.BAS from drive B to the printer device. The N parameter tells PIP to number each line. The T8 parameter expands tabs to every eighth column. The U parameter translates lower-case letters to upper-case as the file is printed.

```
A>PIP PORTION.TXT=LETTER.TXT[SDear Sir^Z
QSincerely^Z]
```

This command abstracts a portion of the LETTER.TXT file from drive A by searching for the character sequence "Dear Sir" before starting the copy operation. When found, the characters are copied to PORTION.TXT on drive A until the sequence "Sincerely" is found in the source file.

```
B>PIP B:=A:*.COM[VWR]
```

This command copies all files with filetype COM from drive A to drive B. The V parameter tells PIP to read the destination files to ensure that data was correctly transferred. The W parameter lets PIP overwrite any destination files that are marked as RO (Read-Only). The R parameter tells PIP to read files from drive A that are marked with the SYS (System) attribute.

The PUT Command

Syntax: PUT CONSOLE {OUTPUT TO} FILE filespec {o} PUT PRINTER {OUTPUT TO} FILE filespec {o} PUT CONSOLE {OUTPUT TO} CONSOLE
PUT PRINTER {OUTPUT TO} PRINTER

Explanation: The PUT command is a transient utility that lets you direct console output or printer output to a file. PUT allows you to direct the system to put console output or printer output to a file for the next system command or user program entered at the console. Or, PUT directs all subsequent console or printer output to a file when you include the SYSTEM option.

Console output is directed to a file until the program terminates. Then, console output reverts to the console. Printer output is directed to a file until the program terminates. Then printer output is directed back to the printer.

When you use the SYSTEM option, all subsequent console/printer output is directed to the specified file. This option terminates when you enter the PUT CONSOLE or PUT PRINTER command.

The syntax for the option list is

[{ECHO | NO ECHO} {FILTER | NO FILTER} | {SYSTEM}]

Table 5-14 defines the preceding option list.

Table 5-14. PUT Options

Option	*Meaning*
ECHO	specifies that the output is echoed to the console. ECHO is the default option when you direct console output to a file.
NO ECHO	specifies that the file output is not to be echoed to the console.
FILTER	specifies that filtering of control characters is allowed, which means that control characters are translated to

	printable characters. For example, an escape character is translated to ^[.
NO FILTER	means that PUT does not translate control characters. This is the default option.
SYSTEM	specifies that system output and program output is written to the file specified by filespec. Output is written to the file until a subsequent PUT CONSOLE command redirects console output back to the console.

Direct Console Output to a File

Syntax: PUT CONSOLE {OUTPUT} TO FILE filespec {[o]}

Explanation: The preceding form of the PUT command tells the system to direct subsequent console output to a file.

Example:

```
A>PUT CONSOLE OUTPUT TO FILE XOUT [ECHO]
```

The preceding command directs console output to file XOUT with the output echoed to the console.

Put Printer Output to a File

Syntax: PUT PRINTER {OUTPUT TO} FILE filespec {[o]}

Explanation: The preceding form of the PUT command directs printer output to a file.

The options are the same as in the PUT CONSOLE command, except that option NO ECHO is the default for the PUT PRINTER command. Note that if ECHO is specified, printer output is echoed to the printer.

Examples:

```
A>PUT PRINTER OUTPUT TO FILE XOUT
A>MYPROG
```

The preceding example directs the printer output of program MYPROG to file XOUT. The output is not echoed to the printer.

```
A>PUT PRINTER OUTPUT TO FILE XOUT2
[ECHO,SYSTEM]
```

The preceding command directs all printer output to file XOUT2 and to the printer, and the PUT is in effect until you enter a PUT PRINTER OUTPUT TO PRINTER command.

The printer output can be directed to one or more files. The output to these files is terminated when you revert printer output to the printer using the following command:

PUT PRINTER OUTPUT TO PRINTER

Terminate Console Output to a File

Syntax: PUT CONSOLE {OUTPUT TO} CONSOLE

Explanation: The preceding form of the PUT command directs console output to the console.

Example:

```
A>PUT CONSOLE OUTPUT TO CONSOLE
```

The preceding command directs console output to the console.

Terminate Printer Output to a File

Syntax: PUT PRINTER {OUTPUT TO} PRINTER

Explanation: The preceding form of the PUT command directs the printer output to the printer.

Example: A>PUT PRINTER OUTPUT TO PRINTER

The preceding example directs printer output to the printer.

The RENAME Command

Syntax: RENAME {new-filespec=old-filespec}

Explanation: The RENAME command lets you change the name of a file that is cataloged in the directory of a disk. It also lets you change several filenames if you use wildcards in the filespecs. You can abbreviate RENAME to REN.

The new-filespec must not be the name of any existing file on the disk. The old-filespec identifies an existing file or files on the disk.

The RENAME command changes the file named by old-filespec to the name given as new-filespec.

RENAME does not make a copy of the file. RENAME changes only the name of the file.

If you omit the drive specifier, RENAME assumes the file to rename is on the default drive. You can include a drive specifier as a part of the newname. If both file specifications name a drive, it must be the same drive.

If the file given by oldname does not exist, RENAME displays the following message on the screen:

```
No File
```

If the file given by newname is already present in the directory, RENAME displays the following message on the screen:

```
Not renamed: filename.typ file already exists, delete (Y/N)?
```

If you want to delete the old file, type Y to delete. Otherwise, type N to keep the old file and not rename the new file.

If you use wildcards in the filespecs, the wildcards in the new filespec must correspond exactly to the wildcards in the old filespec. For example, in the following two commands, the wildcard filespecs correspond exactly:

```
A>REN *.TX1=*.TEX A>REN A*.T*=S*.T*
```

In the following example, the wildcards do not match and CP/M Plus returns an error message.

```
A>REN A*.TEX=A.T*
```

Examples:

```
A>RENAME NEWASM.BAS=OLDFILE.BAS
```

The file OLDFILE.BAS changes to NEWASM.BAS on drive A.

```
A>RENAME
```

The system prompts for the filespecs:

```
Enter New Name:X.PRN Enter Old Name:Y.PRN
Y      .PRN=X      .PRN
A>
```

File Y.PRN is renamed X.PRN on drive A.

```
B>REN A:X.PAS = Y.PLI
```

The file Y.PLI changes to X.PAS on drive A.

```
A>RENAME S*.TEX=A*.TEX
```

The preceding command renames all the files matching the wildcard A*.TEX to files with filenames matching the wildcard S*.TEX, respectively.

```
A>REN B:NEWLIST=B:OLDLIST
```

The file OLDLIST changes to NEWLIST on drive B. Because the second drive specifier, B: is implied by the first, it is unnecessary in this example. The preceding command line has the same effect as the following:

```
A>REN B:NEWLIST=OLDLIST
```
or

```
A>REN NEWLIST=B:OLDLIST
```

The RMAC Command

Syntax: RMAC filespec {$Rd | $Sd | $Pd}

Explanation: RMAC is a relocatable macro assembler that assembles files of type ASM into REL files that can be linked to create COM files.

The RMAC command options specify the destination of the output files. The additional specifier d defines the destination drive of the output files. A-O specifies drives A through O. X means output to the console, P means output to tne printer, and Z means zero output. Table 5-15 lists the RMAC command options.

Table 5-15. RMAC Command Options

Option		*d=output option*
R	drive for REL file	(A-O, Z)
S	drive for SYM file	(A-O, X, P, Z)
P	drive for PRN file	(A-O, X, P, Z)

In the MAC command, the assembly parameter of H controls the destination of the HEX file. In the RMAC command this parameter is replaced by R, which controls the destination of the REL file; however, you cannot direct the REL file to the console or printer, RX or RP, because the REL file is not an ASCII file.

Examples:

```
A>RMAC TEST $PX SB RB
```

In the preceding example RMAC assembles the file TEST.ASM from drive A, sends the listing file (TEST.PRN) to the console, puts the symbol file (TEST.SYM) on drive B and puts the relocatable object file (TEST.REL) on drive B.

The SAVE Command

Syntax: SAVE

Explanation: The SAVE command copies the contents of memory to a file. To use the SAVE utility, first issue the SAVE command, then run your program which reads a file into memory. When your program exits, it exits to the SAVE utility. The SAVE utility prompts you for the filespec to which the memory is to be copied, and the beginning and ending address of the memory to be saved.

Example:

```
A>SAVE
```

The preceding command activates the SAVE utility. Now enter the name of the program that loads a file into memory.

```
A>SID dump.com
```

Next, execute the program.

```
# g0
```

When the program exits, SAVE intercepts the return to the system and prompts you for the filespec and the bounds of memory to be saved.

```
SAVE Ver 3.0 File (or RETURN to exit)?dump2.com Delete
dump2.com?Y From?100 To?400

A>
```

The contents of memory from 100H, hexadecimal, to 400H is copied to file DUMP2.COM.

The SET Command

Syntax: SET [options] SET d: [options] SET filespec [options]

Explanation: The SET command initiates password protection and time stamping of files in the CP/M Plus system. It also sets file and drive attributes, such as the Read-Only, SYS, and user-definable attributes. It lets you label a disk and password protect the label.

The SET command include options that affect the disk directory, the drive, or a file or set of files. The discussion of the SET command explicitly states which of the three categories are affected.

To enable time stamping of files, you must first run INITDIR to format the disk directory.

Set File Attributes

Syntax: SET filespec [attribute-options]

Explanation: The preceding SET command sets the specified attributes of a file or a group of files.

Table 5-16. SET File Attributes

Option	*Meaning*
DIR	Sets the file from the SYS directory to the (DIR) attribute.
SYS	Gives the file the System SYS attribute.
RO	Sets the file attribute to allow Read-Only access.
RW	Sets the file attribute to allow Read-Write access.
ARCHIVE=OFF	Sets the archive attribute to off. This means that the file has not been backed up (archived). PIP with the [A] option can copy files with the archive attribute set to OFF. PIP with this option requires an ambiguous filespec and copies only files that have been created or changed since the last time they were backed up with the PIP[A] option. PIP then sets the archive attribute to ON for each file successfully copied.
ARCHIVE=ON	Sets the archive attribute to on. This means that the file has been backed up (archived). The archive attribute can be turned on explicitly by the SET command, or it can be turned on by PIP when copying a group of files with the PIP [A] option. The archive attribute is displayed by DIR.
F1=ON|OFF	Turns on or off the user-definable file attribute F1.

F2=ON|OFF	Turns on or off the user-definable file attribute F2.
F3=ON|OFF	Turns on or off the user-definable file attribute F3.
F4=ON|OFF	Turns on or off the user-definable file attribute F4.

Example:

```
A>SET MYFILE.TEX [RO SYS]
```

The preceding command sets MYFILE.TEX to Read-Only and System.

```
A>SET MYFILE.TEX [RW DIR]
```

The preceding command sets MYFILE.TEX to Read-Write with the Directory (DIR) attribute.

Set Drive Attribute

Syntax: SET {d:} [RO]
SET {d:} [RW]

Explanation: The preceding SET commands set the specified drive to Read-Only or Read-Write. If a drive is set to Read-Only, PIP cannot copy a file to it, ERASE cannot delete a file from it, RENAME cannot rename a file on it. You cannot perform any operation that requires writing to the disk. When the specified drive is set to Read-Write, you can read or write to the disk in that drive. If you enter a CTRL-C to the system prompt, all drives are reset to Read-Write.

Example:

```
A>SET B: [RO]
```

The preceding command sets drive B to Read-Only.

Assign a Label to the Disk

Syntax: SET {d:} [NAME=labelname.typ]

Explanation: The preceding SET command assigns a label (name) to the disk in the specified or default drive.

CP/M Plus provides a facility for creating a directory label for each disk. The directory label can be assigned an eight-character name and a three-character type similar to a filename and filetype. Label names make it easier to catalog disks and keep track of different disk directories. The default label name is LABEL.

Example:

```
A>SET [NAME=DISK100]
```

The preceding example labels the disk on the default drive DISK100.

Assign Password to the Label

Syntax: SET [PASSWORD=password]
SET [PASSWORD=<cr>

Explanation: The first form of the preceding SET command assigns a password to the disk label. The second form of the command removes password protection from the label.

You can assign a password to the label. If the label has no password, any user who has access to the SET program can set other attributes to the disk which might make the disk inaccessible to you. However, if you assign a password to the label, then you must supply the password to set any of the functions controlled by the label. SET always prompts for the password if the label is password-protected.

Examples:

```
A>SET          [PASSWORD=SECRET]          A>SET
[PASSWORD=<cr>
```

The first command assigns SECRET to the disk label. The second command nullifies the existing password.

Note: If you use password protection on your disk, be sure to record the password. If you forget the password, you lose access to your disk or files.

Enable/Disable Password Protection for Files on a Disk

Syntax: SET [PROTECT=ON] SET [PROTECT=OFF]

Explanation: The first form of the SET command turns on password protection for all the files on the disk. The password protection must be turned on before you can assign passwords to individual files or commands.

The second SET command disables password protection for the files on your disk.

After a password is assigned to the label and the PROTECT option is turned on, you are ready to assign passwords to your files.

You can always determine if a disk is password-protected by using the SHOW command to display the label.

Assign Passwords to Files

Syntax: SET filespec [PASSWORD=password]

Explanation: The preceding SET command sets the password for filespec to the password indicated in the command tail. Passwords can be up to eight characters long. Lower-case letters are translated to upper-case. You can use wildcards in the filespec. SET assigns the specified password to the files that match the wildcard-filespec.

Note that Locoscript ignores the password protection on files.

Note: always record the passwords that you assign to your files. Without the password, you cannot access those files unless password protection is turned off for the whole disk. If you forget the password to the directory label, you cannot turn off the password protection for the disk.

Example:

```
A>SET MYFILE.TEX [PASSWORD=MYFIL]
```

MYFIL is the password assigned to file MYFILE.TEX.

Set Password Protection Mode for Files with Passwords

Syntax: SET filespec [PROTECT=READ]
SET filespec [PROTECT=WRITE]
SET filespec [PROTECT=DELETE] SET filespec [PROTECT=NONE]

Explanation: You can assign one of four modes of password protection to your file. The protection modes are READ, WRITE, DELETE, and NONE and are described in the following table.

Table 5-17. Password Protection Modes

Mode	*Protection*
READ	The password is required for reading, copying, writing, deleting, or renaming the file.
WRITE	The password is required for writing, deleting, or renaming the file. You do not need a password to read the file.
DELETE	The password is only required for deleting or renaming the file. You do not need a password to read or modify the file.
NONE	No password exists for the file. If a password exists, this modifier can be used to delete the password.

Assign a Default Password

Syntax: SET [DEFAULT=password]

Explanation: The preceding set command assigns a default password for the system to use during your computer session. The system uses the default password to access password-protected files if you do not specify a password, or if you enter an incorrect password. The system lets you access the file if the default password matches the password assigned to the file.

Example:

```
B>SET *.TEX [PASSWORD=SECRET, PROTECT=WRITE]
```

The preceding command assigns the password SECRET to all the TEX files on drive B. Each TEX file is given a WRITE protect mode to prevent unauthorized editing.

Example:

```
A>SET [DEFAULT=dd]
```

The preceding command instructs the system to use dd as a password if you do not enter a password for a password-protected file.

Set Time Stamp Options on Disk

Syntax:

SET [CREATE=ON]
SET [ACCESS=ON]
SET [UPDATE=ON]

Explanation: The preceding SET commands allow you to keep a record of the time and date of file creation and update, or of the last access and update of your files.

Note that Locoscript 1.20 does not update access times and you cannot use the facility with earlier versions.

[CREATE=ON] turns on CREATE time stamps on the disk in the default drive. To record the creation time of a file, the CREATE option must have been turned on before the file is created.

[ACCESS=ON] turns on ACCESS time stamps on the disk in the default drive. ACCESS and CREATE options are mutually exclusive. This means that only one can be in effect at a time. If you turn on the ACCESS time stamp on a disk that has the CREATE time stamp, the CREATE time stamp is automatically turned off.

[UPDATE=ON] turns on UPDATE time stamps on the disk in the default drive. UPDATE time stamps record the time the file was last modified.

To enable time stamping, you must first run INITDIR to format the disk directory for time and date stamping.

Although there are three kinds of date/time stamps, only two date/time stamps can be associated with a given file at one time. You can choose to have either a CREATE date or an ACCESS date for files on a particular disk.

When you set both UPDATE and CREATE time stamps, notice that editing a file changes both the UPDATE and CREATE time stamps. This is because ED does not update the original file but creates a new version with the name of the original file.

Example:

```
A>SET [ACCESS=ON]
```

The DIR with [FULL] option displays the following date and time stamps:

```
B>DIR [FULL]

Directory for Drive B:

    Name         Bytes   Recs  Attributes  Prot    Update   Access
-------------------------------------------------------------------
ONE     .TEX      9k      71   Dir RW      None    10:56    08/03/81
THREE   .TEX     12k      95   Dir RW      None    15:45    08/05/81
```

```
A>SET [CREATE=ON,UPDATE=ON]
```

The following DIR output below shows how files with create and update time stamps are displayed.

```
B>DIR [FULL]

Directory for Drive B:

    Name        Bytes  Recs  Attributes  Prot        Update              Create
-------------------------------------------------------------------------------------
GENLED   .DAT   109k   873   Dir RW      None   08/05/81    14:01   08/01/81  09:36
RECEIPTS.DAT     59k   475   Dir RW      None   08/08/81    12:11   08/01/81  09:40
INVOICES.DAT     76k   608   Dir RW      None   08/08/81    08:46   08/01/81  10:15
```

Additional SET Examples

Examples:

```
A>SET *.COM [SYS,RO,PASS=123,PROT=READ]
```

The preceding setting gives the most protection for all the COM files on drive A. With the password protection mode set to READ, you cannot even read one of the COM files without entering the password 123, unless the default password has been set to 123. Even if the correct password is entered, you still cannot write to the file because the file is Read-Only.

```
A>SET *.COM [RW,PROTECT=NONE,DIR]
```

The preceding command reverses the protection and access attributes of the COM files affected by the previous example. After executing the preceding command, there is no password protection, the files of type COM can be read from or written to, and are set to DIR files.

The SET24X80 Command

Syntax: SET24X80 ON
SET24X80 OFF

Explanation: SET24X80 ON sets 24 x 80 mode. SET24X80 restores the screen to its full size. The full size depends on the machine, the country and whether or not the status line is enabled: see your User Guide.

If the Parameter is omitted then ON is assumed.

The SETDEF Command

Syntax: SETDEF {d:{,d:{,d:{,d:}}}} {[TEMPORARY=d:] | [ORDER=(typ {,typ})]}
SETDEF [DISPLAY | NO DISPLAY]
SETDEF [PAGE | NOPAGE]

Explanation: The SETDEF command lets you display or define the disk search order, the temporary drive, and the filetype search order. The SETDEF definitions affect only the loading of programs and/or execution of SUBMIT (SUB) files. The SETDEF command also lets you turn on/off the DISPLAY and PAGE modes for the system. When DISPLAY mode is on, the system displays the location and name of programs loaded or SUB files executed. When PAGE mode is on, CP/M Plus utilities stop after displaying one full screen of information. Press any key to continue the display.

The system usually searches the specified drive or the default drive for files. The user can use the SETDEF command, to extend the search for program files and submit files, for execution purposes only.

Note: A CP/M Plus program file has a filetype of COM. A file containing commands to be executed by SUBMIT has a filetype of SUB.

Display the Program Loading Search Definitions

Syntax: SETDEF

Explanation: The preceding form of the SETDEF command displays the disk search order, the temporary drive, and the filetype search order.

Assign the Drive for Temporary Files

Syntax: SETDEF [TEMPORARY=D:]

Explanation: The preceding form of the SETDEF command defines the disk drive to be used for temporary files. The default drive used for temporary files is the system default drive.

Example:

```
A>SETDEF [TEMPORARY=B:]
```

The preceding command sets disk drive B as the drive to be used for temporary files.

Define the Disk Drive Search Order

Syntax: SETDEF { d: {,d: {,d: {,d:}}}}

Explanation: The preceding form of the SETDEF command defines the disks to be searched by the system for programs and/or submit files to be executed. The CP/M Plus default is to search only the default drive.

Note: * can be substituted for d: to indicate that the default drive is to be included in the drive search order.

Example:

```
A>SETDEF B:,*
```

The preceding example tells the system to search for a program on drive B, then, if not found, search for it on the default drive.

Define the Filetype Search Order

Syntax: SETDEF [ORDER = (typ {,typ})]
where typ = COM or SUB

Explanation: The preceding form of the SETDEF command defines the filetype search order to be used by system for program loading. The filetype, indicated as typ in the syntax line, must be COM or SUB. The CP/M Plus default search is for COM files only.

Example:

```
A>SETDEF [ORDER=(SUB,COM)]
```

The preceding command instructs the system to search for a SUB file to execute. If no SUB file is found, search for a COM file.

Turn On/Off System Display Mode

Syntax: SETDEF [DISPLAY | NO DISPLAY]

Explanation: The preceding command turns the system display mode on or off. The default system display mode is off. When the display mode is on, CP/M Plus displays the following information about a program file before loading it for execution: drive, filename, filetype (if any), and user number (if not the default user number).

Example:

```
A>SETDEF [DISPLAY]
```

The preceding command turns on the system display mode. The system now displays the name and location of programs loaded or submit files executed. For example, if you enter the PIP command after turning on the system display mode, CP/M Plus displays the following:

```
A>PIP
A:PIP          COM
CP/M Plus PIP VERSION 3.0
*
```

indicating that the file PIP.COM was loaded from drive A under the current user number. If the current user number is not 0, and if PIP.COM does not exist under the current user number, then the system displays the location of PIP.COM as follows:

```
4A>PIP
A:PIP          COM (User 0)
CP/M Plus PIP VERSION 3.0
*
```

indicating that PIP.COM was loaded from drive A under user number 0. This mode is in effect until you enter

```
SETDEF [NO DISPLAY]
```

to turn off the system DISPLAY mode.

Turn On/Off System Page Mode

Syntax: SETDEF [PAGE| NO PAGE]

Explanation: The preceding command turns on/off the system page mode. When the PAGE mode is set to on, CP/M Plus utilities stop after displaying one full screen of information, called a console page. The utilities resume after you press any key.

The default setting of the system page mode is ON.

Example:

```
A>SETDEF [NO PAGE]
```

The preceding command turns off the system page mode. CP/M Plus utilities do not pause after displaying a full console page, but continue to scroll.

The SETKEYS Command

Syntax: SETKEYS filename

The command file contains the keyboard configuration data and has the following syntax.

Each line contains a key definition or an expansion token definition. A key definition associates a key in a given shift state, or states, with a character or token value. An explanation token definition associated an expansion token with a string.

Key Definition

A key definition consists of a key number optionally followed by a shift state or states, followed by the required character in quotes. Any other characters on the same line are treated as comment.

```
71      "z" lower case Z
71 S    "Z" upper case Z
```

The shift states are machine dependent:

CPC6128
- S for shift
- C for control
- N for nothing to indicate no shift

PCW8256
- S for shift
- A for alt
- E for extra
- SA for shift and alt
- N or nothing to indicate no shift

More than one shift state may be given in which case the definition will apply to all the shift states.

The character associated with thekey is either the character itself or an escape sequence. .

Characters in the range #20..#FF other than ^ or " stand for themselves. ^ introduces an escape sequence. Control codes must be represented by escape sequences.

^ followed by a character in the range #40..#FF masks the character with #1F thus ^A gives Control A.

^^ gives the character ^

^" gives the character "

^ followed by a number enclosed by single quotes gives a character of that value. ^'#D' gives carriage return.

^ followed by the name of a control code enclosed by single quotes gives that control code, ^'ESC' gives the ESC code.

The names of the control codes are NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT, FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US, SP, DEL, XON, XOFF.

Examples

```
67 "q" lower case Q
67 S "Q" upper case Q
67 C "^Q" control Q

66 N S C "^'ESC'" escape is always escape
```

Expansion Token Definitions

An expansion token definition consists of E followed by the token number followed by the expansion string enclosed by "", followed by a comment it required. The characters in the string are represented by the same characters and escape sequences as in key definitions.

Token numbers are machine dependent:

CPC6128: #80..#9F.
PCW8256: #80..#9E. (#9F is the ignore token)

Any key can be set to one of these values in which case it will return the expansion string.

Examples

```
E #80 "DIR B: [SIZE]M"
E #87 "MALLARD^M"
```

In an expansion token definition bit 7 of the token value is ignored.

When parsing the command file any line which contains an error is displayed on the console with an error message and ignored, parsing continues withe the next line.

Details of using the SETKEYS command are given in your User Guide.

The SETLST Command

Syntax: SETLST filename

The command file contains the information to send to the LST: device. All the information is represented by characters in the range #20..#FF. Control codes must be used as follows.

Characters in the range #20..#FF other than ^ stand for themselves. ^ introduces an escape sequence.

^ followed by a character in the range #40..#FF masks the character with #1F thus ^A gives Control A.

^^ gives the character ^

^ followed by a number enclosed by single quotes gives a character of that value. ^'#D' gives carriage return.

^ followed by the name of a control code enclosed by single quotes gives that control code, ^'ESC' gives the ESC code.

The names of the control characters are NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT, FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US, SP, DEL, XON, XOFF.

Any illegal escape sequence will produce an error message and the sequence will be ignored.

Examples

```
^'ESC'A5563
^Abcd
^'STX'^'#F4
```

Details of using the SETLST command are given in your User Guide.

The SETSIO Command

Syntax: SETSIO option option option ...

Where an option is any, or all, of the following in any order:

Baud rate

TX 300	sets transmitter baud rate to 300 baud
RX 134.5	sets receiver baud rate to 134.5 baud
9600	sets both baud rates to 9600 baud

The baud rate must be one of 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200.

Number of data bits

BITS 7	7 data bits

The number of data bits must be one of 5, 6, 7 or 8.

Number of stop bits

STOP 1	1 stop bit

The number of stop bits must be one of 1, 1.5 or 2.

Parity

PARITY EVEN	sets even parity
PARITY ODD	sets odd parity
PARITY NONE	sets no parity

Interrupts

INTERRUPT ON INTERRUPT OFF	Determines whether the SIO device buffers characters on interrupt or not (version 1.4 or greater)

XON Protocol

XON ON	enables XON protocol
XON OFF	disables XON protocol

Control signal handshake

HANDSHAKE ON	enabled handshake
HANDSHAKE OFF	disables handshake

For the words TX, RX, STOP, BITS, PARITY, XON and HANDSHAKE only the initial letter is required, the remainder of the word is ignored.

Explanation: The SIO is initialised as required. For baud rates and the XON options the CP/M Plus character I/O table entry will be changed. The SIO is device number.

The baud rate option without a preceding RX or TX sets both baud rates.

If the baud rate is specified but the number of stop bits is not specified then the number of stop bits is set to 1 if the baud rate is greater than 110 otherwise it is set to 2. Any other option not specified retains its original value.

If an option is given more than once, or if both baud rate and TX baud rate etc are given then the later option in the list is used. For example SETSIO 9600, 300 will set the baud rate to 300.

Any illegal option will produce an error message and the option will be ignored.

After the SIO has been initialised the current state of the SIO settings is displayed in the same form as the command as follows:

9600 Bits 8 Parity none Stop 1 Xon off Handshake on

Thus the command SETSIO may be used to examine the current settings.

Further details of using SETSIO are given in your User Guide.

Examples

```
SETSIO PARITY EVEN
SETSIO
SETSIO 9600, P NONE, HANDSHAKE=ON,STOP 2, BITS
5
```

The SHOW Command

Syntax: SHOW {d:}{[SPACE |LABEL |USERS |DIR |DRIVE]}

Explanation: The SHOW command displays the following disk drive information:

- access mode and amount of free disk space

- disk label
- current user number
- number of files for each user number on the disk
- number of free directory entries for the disk
- drive characteristics

Display Access Mode and Disk Space Available

Syntax: SHOW {d:}{[SPACE]}

Explanation: The preceding form of the SHOW command displays the drive, the access mode for that drive, and the remaining space in kilobytes for the specified drive. SHOW by itself displays the information for all logged-in drives in the system.

Examples:

```
A>SHOW B:
B: RW, Space:      88k
A>SHOW
A: RW, Space:       4k
B: RW, Space:      88k
```

The first example shows that drive B has Read-Write access and it has 88K bytes of space left. The second example shows that drive A also is Read-Write and has only 4K bytes left and drive B is Read-Write and has 88K bytes left.

Display Disk Label

Syntax: SHOW {d:}[LABEL]

Explanation: The preceding form of the SHOW command displays disk label information.

Example:

```
A>SHOW B:[LABEL]
```

The preceding command displays the following for drive B:

```
Label for drive B:
Directory      Passwds     Stamp    Stamp     Label Created       Label Updated
Label          Reqd        Create   Update
---------------------------------------------------------------------------------
TOMSDISK.        on          on       on      07/04/85  10:30  07/08/85  09:30
```

The first column, directory label, displays the name assigned to that drive directory. The second column, Passwds Reqd, shows that password protection has been turned on for that drive.

As described in the SET command, each file can have up to two time stamps. The first of these time stamps can be either the creation date and time for the file or the date and time of the last access to the file. Access is defined as reading from or writing to the file. The third column of the SHOW [LABEL] output displays both the type of stamps and whether or not it is on. In the preceding example, creation time stamps are given to new files as shown by the stamp create column heading.

The fourth column displays the status of the second time stamp field, the update time stamp. Update time stamps display the date and time of the last update to a file, that is, the last time someone wrote to the file. In the SHOW [LABEL] display, update time stamps are turned on.

Besides showing the password protection and the active time stamps on a drive, SHOW [LABEL] also displays the date and time that the label was created and last updated.

Display User Number Information

Syntax: SHOW {d:}[USERS]

Explanation: The preceding command displays the current user number and all the users on the drive and the corresponding number of files assigned to them.

Example:

```
A>SHOW [USERS] Active User :  0
Active Files:  0
A: # of files:  28
A: Number of free directory entries: 34
A>
```

Display Number of Free Directory Entries

Syntax: SHOW {d:}[DIR]

Explanation: The preceding command displays the number of free directory entries on the specified drive.

Example:

```
A>SHOW B:[DIR]

B: Number of free directory entries: 24

A>
```

The preceding command shows that there are 24 free directory entries on drive B.

Display Drive Characteristics

Syntax: SHOW {d:}[DRIVE]

Explanation: The preceding form of the SHOW command displays the drive characteristics of the specified drive.

Example:

```
A> SHOW [DRIVE]
```

The following is an example of the system display for the preceding command:

```
    A: Drive Characteristics
1,368: 128 Byte Record Capacity
  171: Kilobyte Drive Capacity
   64: 32 Byte Directory Entries
```

```
 64: Checked Directory Entries
128: Records / Directory Entry
  8: Records / Block
 36: Sectors / Track
512: Bytes / Physical Record
```

The SID Command

Syntax: SID {pgm-filespec} {,sym-filespec}

Explanation: The SID (Symbolic Instruction Debugger) allows you to monitor and test programs developed for the 8080 microprocessor. SID supports real-time breakpoints, fully monitored execution, symbolic disassembly, assembly, and memory display and fill functions. Utility programs are supplied with CP/M Plus that can be dynamically loaded with SID to provide traceback and histogram facilities.

SID commands display memory and CPU registers and direct the breakpoint operations during the debugging session.

Without a file specification SID loads into memory without a test program. Use this form to examine memory or to write and test simple programs using the A command. You must not use the SID commands G, T, or U, described later, until you have first loaded a test program.

A SID command line with a pgm-filespec loads both SID and the test program into memory. If the filetype is omitted from the filespec, COM is assumed. SID optionally loads in a symbol table file specified by sym-filespec. The sym-filespec needs no filetype because SID looks for a file with filetype SYM. Use the C, G, T, or U command to begin execution of the test program under supervision of SID.

Use CTRL-S to halt the screen display. CTRL-Q restarts the display. Abort lengthy displays by typing any keyboard character. Use CTRL-C to exit from SID.

SID can address absolute memory locations through symbolic expressions. A symbolic expression evaluates to either an address or a data item.

A symbolic expression can be a name from a SYM file produced from your program by a CP/M Macro Assembler. When you precede the symbolic expression with a period, SID returns its address in hexadecimal. When you precede the symbolic expression with the at sign, @, SID returns the 16-bit value stored at that location and the next contiguous location. When you precede the symbolic expression with an equal sign, SID returns the 8-bit value stored at that location. For two-byte expressions, this is the low byte because the 8080 microprocessor stores the low value of a two-byte word first.

A symbolic expression can be a literal value in hex, decimal, or ASCII, as indicated in the following list:

- SID uses literal hex values as given, but truncates any digits in excess of four on the left. The leftmost digit is the most significant digit. The rightmost digit is the least significant digit.

- To indicate decimal values precede them with a hash sign, #. Decimal values that evaluate to more than four hex digits are evaluated as the modulo of hex value FFFF. For example, #65534=FFFEH, while #65536=0001H.

- SID translates literal ASCII character strings between apostrophes to the hex value of the two rightmost ASCII characters.

You can combine symbolic expressions with the symbolic operators, + or -, to produce another symbolic expression. Symbolic expressions combined in this way can be used to calculate the offset of an indirectly addressed data item, for example a subscripted variable. A special up-arrow operator, ^, can reference the top-of-stack item. A string of n ^ operators can reference the nth stack item without changing stack content or the stack pointer.

Table 5-18 lists the SID commands with their corresponding parameters and options. The actual command letter is printed in boldface. The parameters are in lower-case and follow the command letter. Optional items are in braces. Replace the arguments with the appropriate symbolic expressions as listed. Where two symbolic expressions are needed, SID can calculate the second one from the first using the symbolic operators described previously.

Table 5-18. SID Commands

Name	*Syntax*	*Meaning*
Assemble	As	Enter assembly language statements. s is the start address.
Call	Cs {b{,d}}	Call to memory location from SID. s is the called address, b is the value of the BC register pair, and d is the value of the DE register pair.
Display	D {W}{s}{,f}	Display memory in hex and ASCII. W specifies a 16-bit word format, s is the start address, and f is the finish address.
Load	E pgm-filespec {,sym-filespec}	Load program and symbol table for execution.
Load	E* sym-filespec	Load a symbol table file.
Fill	F s,f,d	Fill memory with constant value. s is the start address, f is the finish address, and d is an 8-bit data item.

Table 5-18 (continued)

Go	G {p}{,a{,b}}	Begin execution. p is a start address, a is a temporary breakpoint, and b is a second temporary breakpoint. Go exits SID by performing a warm boot.
Hex	H Ha Ha,b	Displays all symbols with addresses in hex, decimal, and ASCII values of a. The second syntax performs number and character conversion, where a is a symbolic expression, and the third syntax computes hex sum and difference of a and b, where a and b are symbolic expressions.
Input	I command tail	Input CCP command line.
List	L {s}{,f}	List 8080 mnemonic instructions. s is the start address, and f is the finish address.
Move	M s,h,d	Move memory block. s is the start address, h is the high address of the block, and d is the destination start address.
Pass	P {p{,c}}	Pass point set, reset, and display. p is a permanent breakpoint address, and c is initial value of pass counter.
Read	R filespec{,d}	Read code/symbols. d is an offset to each address.
Set	S {W}s	Set memory values. s is an address where value is sent, W is a 16-bit word.
Trace	T {n{,c}}	Trace program execution. n is the number of program steps, and c is the utility entry address.
Trace	T {W}{n{,c}}	Trace without call. W instructs SID not to trace subroutines, n is the number of program steps, and c is the utility entry address.

Untrace	U {W}{n{,c}}	Monitor execution without trace. n is the number of program steps, c is the utility entry address, W instructs SID not to trace subroutines.
Value	V	Display the value of the next available location in memory (NEXT), the next location after the largest file read in (MSZE), the current value of the program counter (PC), and the address of the end of available memory (END).
Write	W filespec{,s,f}	Write the contents of a contiguous block of memory to filespec. s is the start address, f is the finish address.
Examine	X {f}{r}	Examine/alter CPU state. f is flag bit C, E, I, M, or Z; r is register A, B, D, H, P or S.

Examples:

```
A>SID
```

In the preceding example CP/M Plus loads SID from drive A into memory. SID displays the # prompt when it is ready to accept commands.

```
A>B:SID SAMPLE.HEX
```

In the preceding example, CP/M Plus loads SID and the program file SAMPLE.HEX into memory from drive B. SID displays:

```
NEXT    MSZE    PC    END
nnnn   mmmm    pppp    eeee
```

In the preceding example, nnnn is a hexadecimal address of the next free location following the loaded program, and mmmm is the next location after the largest program. This is initially the same value as NEXT. pppp is the initial hexadecimal value of the the program counter. eeee is the hexadecimal address of the logical end of the TPA.

```
#DFE00+#128+5
```

In the preceding example the first pound sign, #, is the SID prompt. This SID command, D, displays the values stored in memory starting at address FE80 (FE00 + #128) and ending at address FE85 (FE80 + 5).

SID Utilities

The SID utilities HIST.UTL and TRACE.UTL are special programs that operate with SID to provide additional debugging facilities. The mechanisms for system initialization, data collection, and data display are described in the CP/M SID™ *Symbolic Instruction Debugger User's Guide*. The following discussion illustrates how a utility is activated. You load the utility by naming it as a parameter when invoking SID:

SID filename.UTL

In the preceding example filename is the name of the utility. Following the initial sign-on, the utility can prompt you for additional debugging parameters.

The HIST utility creates a histogram (bar graph) showing the relative frequency of execution of code within selected program segments of the test program. The HIST utility allows you to monitor those sections of code that execute most frequently.

Upon start-up HIST prompts

```
TYPE HISTOGRAM BOUNDS
```

Enter the bounds in the following format:

aaaa,bbbb

for a histogram between locations aaaa and bbbb inclusive. Collect data in U or T mode, then display results.

The TRACE utility obtains a traceback of the instructions that led to a particular breakpoint address in a program under test. You can collect the addresses of up to 256 instructions between pass points in U or T modes.

The SUBMIT Command

Syntax: SUBMIT {filespec} {argument} ... {argument}

Explanation: The SUBMIT command lets you execute a group or batch of commands from a SUB file, which is a file with filetype of SUB.

Usually, you enter commands one line at a time. If you must enter the same sequence of commands several times, you might find it easier to batch the commands together using the SUBMIT command. To do this, create a file and enter your commands in this file. The file is identified by the filename, and must have a filetype of SUB. When you issue the SUBMIT command, SUBMIT reads the file named by the filespec and prepares it for interpretation by CP/M Plus. When the preparation is complete, SUBMIT sends the file to CP/M Plus line by line, as if you were typing each command.

The SUBMIT command executes the commands from a SUB file as if you are entering the commands from the keyboard.

You create the SUB file with the ED utility. It can contain CP/M Plus commands, nested SUBMIT commands, and input data for a CP/M Plus command or a program.

You can pass arguments to SUB files when you execute them. Each argument you enter is assigned to a parameter in the SUB file. The first argument replaces every occurrence of $1 in the file, the second argument replaces parameter $2, etc., up to parameter $9. For example, if your file START.SUB contains the following commands:

```
ERA $1.BAK
DIR $1
PIP $1=A:$2.COM
```

and you enter the following SUBMIT command:

```
A>SUBMIT START SAM TEX
```

the argument SAM is substituted for every $1 in the START.SUB file, and TEX for every occurrence of $2 in the START.SUB file. SUBMIT then creates a file with the parameter substitutions and executes this file. This file now contains the following commands:

```
ERA SAM.BAK
DIR SAM
PIP SAM=A:TEX.COM
```

If you enter fewer arguments in the SUBMIT command than parameters in the SUB file, the remaining parameters are not included in the commands.

If you enter more arguments in the SUBMIT command than parameters in the SUB file, the remaining arguments are ignored.

To include an actual dollar sign, $ in your SUB file, type two dollar signs, $$. SUBMIT replaces them with a single dollar sign when it substitutes an argument for a parameter in the SUB file. For example, if file AA.SUB contains line:

```
MAC $1 $$$2
```

and you enter the following SUBMIT command:

```
A>SUBMIT AA ZZ SZ
```

then the translated file contains the following:

```
MAC ZZ $SZ
```

Program Input Lines in a SUB File

A SUB file can contain program input lines. Any program input is preceded by a less than sign, <, as in:

```
PIP
<B:=*.ASM
<CON:=DUMP.ASM
< DIR
```

The three lines after PIP are input lines to the PIP command. The third line consists only of the < sign, indicating a carriage return. The carriage return causes PIP to return to the system to execute the final DIR command.

If the program terminates before using all of the input, SUBMIT ignores the excess input lines and displays the following warning message:

```
Warning: Program input ignored
```

If the program requires more input than is in the SUB file, it expects you to enter the remaining input from the keyboard.

You can enter control characters in a SUB file by using the usual convention of preceding the control character by an up-arrow character, ↑, followed by the letter to be converted to a control character. To enter an actual ↑ character, use the combination ↑↑. This combination translates to a single ↑ in the same manner that $$ translates to a single $.

The SUB File

The SUB file can contain the following types of lines:

- Any valid CP/M Plus command
- Any valid CP/M Plus command with SUBMIT parameters

- Any data input line
- Any program input line with parameters ($0 to $9)

CP/M Plus command lines cannot exceed 128 characters.

Example: The following lines illustrate the variety of lines that can be entered in a SUB file:

```
DIR
DIR *.BAK
MAC $1 $$$4
PIP LST:=$1.PRN[T$2 $3 $5]
DIR *.ASM
PIP <B:=*.ASM
<CON:=DUMP.ASM
< DIR B:
```

Executing the SUBMIT Command

Syntax:

```
SUBMIT
SUBMIT filespec
SUBMIT filespec argument ... argument
```

If you enter only SUBMIT, the system prompts for the rest of the command. You enter the filespec and arguments.

Example:

```
A>SUBMIT
```

The system displays the following prompt. Enter filespec and arguments here, such as:

```
Enter File to Submit: START B TEX
```

Another example could be

```
A>SUBMIT SUBA
```

Still another example using parameters is

```
A>SUBMIT AA ZZ SZ
```

where AA is the SUB file AA.SUB, ZZ is the argument to replace any occurrences of $1 in the AA.SUB file and SZ is the argument to replace all occurrences of $2 in the AA.SUB file.

The PROFILE.SUB Start-up File

Every time you turn on or reset your computer, CP/M Plus automatically looks for a special SUB file named PROFILE.SUB to execute. If it does not exist, then CP/M Plus resumes normal operation. If the PROFILE.SUB file exists, the system executes the commands in the file. This file is convenient to use if you regularly execute a set of commands before you do your regular session on the computer. For example, if you want to be sure that you always enter the current date and time on your computer before you enter any other commands, you can create the PROFILE.SUB file, with ED, and enter the DATE command as follows:

```
DATE SET
```

Then, whenever you bring up the system, the system executes the DATE command and prompts you to enter the date and time. By using this facility, you can be sure to execute a regular sequence of commands before starting your usual session.

The TYPE Command

Syntax: TYPE {filespec {[PAGE]|[NO PAGE]}}

Explanation: The TYPE command displays the contents of an ASCII character file on your screen. The PAGE option displays the console listing in paged mode, which means that the console listing stops automatically after listing n lines of text, where n is usually the system default of 24 lines per page. (See the DEVICE command to set n to a different value.) Press any character to continue listing another n lines of text. Press CTRL-C to exit back to the system. PAGE is the default mode.

The NO PAGE option displays the console listing continuously.

If you do not enter a file specification in the TYPE command the system prompts for a filename with the message:

```
Enter filename:
```

Respond with the filespec of the file you want listed.

Tab characters occurring in the file named by the file specification are expanded to every eighth column position of your screen.

At any time during the display, you can interrupt the listing by pressing CTRL-S. Press CTRL-Q to resume the listing.

Press CTRL-C to exit back to the system.

Make sure the file specification identifies a file containing character data.

If the file named by the file specification is not present on the specified drive, TYPE displays the following message on your screen:

```
No File
```

To list the file at the printer and on the screen, type a CTRL-P before entering the TYPE command line. To stop echoing console output at the printer, type a second CTRL-P. The type command displays the contents of the file until the screen is filled. It then pauses until you press any key to continue the display.

Examples:

```
A>TYPE MYPROG.PLI
```

This command displays the contents of the file MYPROG.PLI on your screen twenty-four lines at a time.

```
A>TYPE B:THISFILE [NO PAGE]
```

This command continuously displays the contents of the file THISFILE from drive B on your screen.

The USER Command

Syntax: USER {number}

Explanation: The USER command sets the current user number. When you start CP/M Plus, 0 is the current user number. You can use a USER command to change the current user number to another in the range 0-15.

CP/M Plus identifies every file with a user number. In general, you can access only files identified with the current user number. However, if you mark a file in user 0 with the SYS attribute, the file can be accessed from all other user numbers.

Note that Locoscript treats CP/M Plus files in user numbers 8 to 15 as limbo files.

Examples:

```
A>USER
```

The system command prompts for the user number, as follows:

```
Enter User#:5 5A>
```

The current user number is now 5 on drive A.

```
A>USER 3 3A>
```

This command changes the current user number to 3.

The XREF Command

Syntax: XREF {d:} filename {$P}

Explanation: The XREF command provides a cross-reference summary of variable usage in a program. XREF requires the PRN and SYM files produced by MAC or RMAC for the program.

The SYM and PRN files must have the same filename as the filename in the XREF command tail. XREF outputs a file of type XRF.

Examples:

```
A>XREF b:MYPROG
```

In this example, XREF is on drive A. XREF operates on the file MYPROG.SYM and MYPROG.PRN which are on drive B. XREF produces the file MYPROG.XRF on drive B.

```
A>XREF b:MYPROG $P
```

In the preceding example, the $P option directs output to the printer.

Section 6

ED: the CP/M Plus Context Editor

Introduction to ED

To do almost anything with a computer you need some way to enter data, a way to give the computer the information you want it to process. The programs most commonly used for this task are called editors. They transfer your keystrokes at the keyboard to a disk file. CP/M Plus's editor is named ED. Using ED, you can easily create and alter CP/M Plus text files.

The correct command format for invoking the CP/M Plus editor is given in "Starting ED." After starting ED, you issue commands that transfer text from a disk file to memory for editing. "ED Operation," details this operation and describes the basic text transfer commands that allow you to easily enter and exit the editor.

"Basic Editing Commands," details the commands that edit a file. "Combining ED Commands," describes how to combine the basic commands to edit more efficiently. Although you can edit any file with the basic ED commands, ED provides several more commands that perform more complicated editing functions, as described in "Advanced ED Commands."

During an editing session, ED can return two types of error messages. "ED Error Messages," lists these messages and provides examples that indicate how to recover from common editing error conditions.

Starting ED

Syntax: ED input-filespec {d: | output-filespec}

To start ED, enter its name after the CP/M Plus prompt. The command ED must be followed by a file specification, one that contains no wildcard characters, such as:

```
A>ED MYFILE.TEX
```

The file specification, MYFILE.TEX in the preceding example, specifies a file to be edited or created. The file specification can be preceded by a drive specification, but a drive specification is unnecessary if the file to be edited is on your default drive. Optionally, the file specification can be followed by a drive specification, as shown in the following example:

```
A>ED MYFILE.TEX B:
```

In response to this command, ED opens the file to be edited, MYFILE.TEX, on drive A, but sends all the edited material to a file on drive B.

Optionally, you can send the edited material to a file with a different filename, as in the following example:

```
A>ED MYFILE.TEX YOURFILE.TEX
```

The file with the different filename cannot already exist or ED prints the following message and terminates.

```
Output File Exists, Erase It
```

The ED prompt, *, appears at the screen when ED is ready to accept a command, as follows

```
A>ED MYFILE.TEX
      : *
```

If no previous version of the file exists on the current disk, ED automatically creates a new file and displays the following message:

```
NEW FILE
      : *
```

Note: before starting an editing session, use the SHOW command to check the amount of free space on your disk. Make sure that the unused portion of your disk is at least as large as the file you are editing, or larger if you plan to add characters to the file. When ED finds a disk or directory full, ED has only limited recovery mechanisms. These are explained in "ED Error Messages."

ED Operation

With ED, you change portions of a file that pass through a memory buffer. When you start ED with one of the preceding commands, this memory buffer is empty. At your command, ED reads segments of the source file, for example MYFILE.TEX, into the memory buffer for you to edit. If the file is new, you must insert text into the file before you can edit. During the edit, ED writes the edited text onto a temporary work file, MYFILE.$$$.

When you end the edit, ED writes the memory buffer contents to the temporary file, followed by any remaining text in the source file. ED then changes the name of the source file from MYFILE.TEX to MYFILE.BAK, so you can reclaim this original material from the back-up file if necessary. ED then renames the temporary file, MYFILE.$$$, to MYFILE.TEX, the new edited file. The following figure illustrates the relationship between the source file, the temporary work file, and the new file.

Note: when you invoke ED with two filespecs, an input file and an output file, ED does not rename the input file to type BAK; therefore, the input file can be Read-Only or on a write-protected disk if the output file is written to another disk.

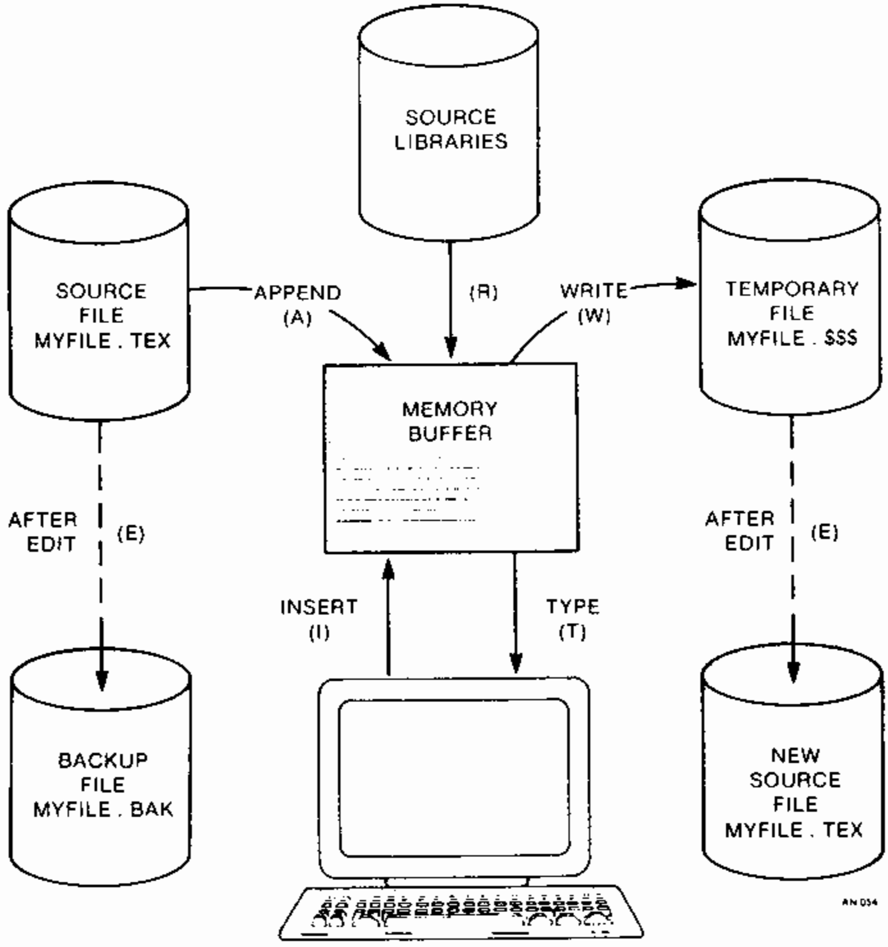


Figure 6-1. Overall ED Operation

In the preceding figure, the memory buffer is logically between the source file and the temporary work file. ED supports several commands that transfer lines of text between the source file, the memory buffer, and the temporary, and eventually final, file. The following table lists the three basic text transfer commands that allow you to easily enter the editor, write text to the temporary file, and exit the editor.

Table 6-1. Text Transfer Commands

Command	*Result*
nA	Append the next n unprocessed source lines from the source file to the end of the memory buffer.
nW	Write the first n lines of the memory buffer to the temporary file free space.
E	End the edit. Copy all buffered text to the temporary file, and copy all unprocessed source lines to the temporary file. Rename files.

Appending Text into the Buffer

When you start ED and the memory buffer is empty, you can use the A (append) command to add text to the memory buffer.

Note: ED can number lines of text to help you keep track of data in the memory buffer. The colon that appears when you start ED indicates that line numbering is turned on. Type -V after the ED prompt to turn the line number display off. Line numbers appear on the screen but never become a part of the output file.

The V (Verify Line Numbers) Command

The V command turns the line number display in front of each line of text on or off. The V command also displays the free bytes and total size of the memory buffer. The V command takes the following forms:

V, -V, 0V

Initially, the line number display is on. Use -V to turn it off. If the memory buffer is empty, or if the current line is at the end of the memory buffer, ED represents the line number as five blanks. The 0V command prints the memory buffer statistics in the form:

```
free/total
```

where free is the number of free bytes in the memory buffer, and total is the size of the memory buffer. For example, if you have a total of 48,253 bytes in the memory buffer and 46,652 of them are free, the 0V command displays this information as follows

```
46652/48253
```

If the buffer is full, the first field, which indicates free space, is blank.

The A (Append) Command

The A command appends, copies, lines from an existing source file into the memory buffer. The A command takes the following form:

```
nA
```

where n is the number of unprocessed source lines to append into the memory buffer. If a hash sign, #, is given in place of n, then the integer 65,535 is assumed. Because the memory buffer can contain most reasonably sized source files, it is often possible to issue the command #A at the beginning of the edit to read the entire source file into memory.

When n is 0, ED appends the unprocessed source lines into the memory buffer until the buffer is approximately half full. If you do not specify n, ED appends one line from the source file into the memory buffer.

ED Exit

You can use the W (Write) command and the E (Exit) command to save your editing changes. The W command writes lines from the memory buffer to the new file without ending the ED session. An E command saves the contents of the buffer and any unprocessed material from the source file and exits ED.

The W (Write) Command

The W command writes lines from the buffer to the new file. The W command takes the form:

```
nW
```

where n is the number of lines to be written from the beginning of the buffer to the end of the new file. If n is greater than 0, ED writes n lines from the beginning of the buffer to the end of the new file. If n is 0, ED writes lines until the buffer is half empty. The 0W command is a convenient way of making room in the memory buffer for more lines from the source file. If the buffer is full, you can use the 0W command to write half the contents of the memory buffer to the new file. You can use the #W command to write the entire contents of the buffer to the new file. Then you can use the 0A command to read in more lines from the source file.

Note: after a W command is executed, you must enter the H command to reedit the saved lines during the current editing session.

The E (Exit) Command

An E command performs a normal exit from ED. The E command takes the form:

```
E
```

followed by a carriage return.

When you enter an E command, ED first writes all data lines from the buffer and the original source file to the $$$ file. If a BAK file exists, ED deletes it, then renames the original file with the BAK filetype. Finally, ED renames the $$$ file from filename.$$$ to the original filetype and returns control to the operating system.

The operation of the E command makes it unwise to edit a back-up file. When you edit a BAK file and exit with an E command, ED erases your original file because it has a BAK filetype. To avoid this, always rename a back-up file to some other filetype before editing it with ED.

Note: any command that terminates an ED session must be the only command on the line.

Basic Editing Commands

The text transfer commands discussed previously allow you to easily enter and exit the editor. This section discusses the basic commands that edit a file.

ED treats a file as a long chain of characters grouped together in lines. ED displays and edits characters and lines in relation to an imaginary device called the character pointer (CP). During an edit session, you must mentally picture the CP's location in the memory buffer and issue commands to move the CP and edit the file.

The following commands move the character pointer or display text in the vicinity of the CP. These ED commands consist of a numeric argument and a single command letter and must be followed by a carriage return. The numeric argument, n, determines the number of times ED executes a command; however, there are four special cases to consider in regard to the numeric argument:

- If numeric argument is omitted, ED assumes an argument of 1.
- Use a negative number if the command is to be executed backwards through the memory buffer. The B command is an exception.
- If you enter a hash sign, #, in place of a number, ED uses the value 65,535 as the argument. A pound sign argument can be preceded by a minus sign to cause the command to execute backwards through the memory buffer, -#.
- ED accepts 0 as a numeric argument only in certain commands. In some cases, 0 causes the command to be executed approximately half the possible number of times, while in other cases it prevents the movement of the CP.

The following table alphabetically summarizes the basic editing commands and their valid arguments.

Table 6-2. Basic Editing Commands

Command	*Action*
B, -B	Move CP to the beginning (B) or end (-B) of the memory buffer.
nC, -nC	Move CP n characters forward (nC) or backward (-nC) through the memory buffer.
nD, -nD	Delete n characters before (-nD) or after (nD) the CP.
I	Enter insert mode.
Istring CTRL-Z	Insert a string of characters.
nK, -nK	Delete (kill) n lines before the CP (-nK) or after the CP (nK).
nL, -nL	Move the CP n lines forward (nL) or backward (-nL) through the memory buffer.
nT, -nT	Type n lines before the CP (-nT) or after the CP (nT).
n, -n	Move the CP n lines before the CP (-n) or after the CP (n) and display the destination line.

The following sections discuss ED's basic editing commands in more detail. The examples in these sections illustrate how the commands affect the position of the character pointer in the memory buffer. Later examples in "Combining ED Commands," illustrate how the commands appear at the screen. For these sections, however, the symbol ^ in command examples represents the character pointer, which you must imagine in the memory buffer.

Moving the Character Pointer

This section describes commands that move the character pointer in useful increments but do not display the destination line. Although ED is used primarily to create and edit program source files, the following sections present a simple text as an example to make ED easier to learn and understand.

The B (Beginning/Bottom) Command

The B command moves the CP to the beginning or bottom of the memory buffer. The B command takes the following forms:

B, -B

-B moves the CP to the end or bottom of the memory buffer; B moves the CP to the beginning of the buffer.

The C (Character) Command

The C command moves the CP forward or backward the specified number of characters. The C command takes the following forms:

nC, -nC

when n is the number of characters the CP is to be moved. A positive number moves the CP towards the end of the line and the bottom of the buffer. A negative number moves the CP towards the beginning of the line and the top of the buffer. You can enter an n large enough to move the CP to a different line. However, each line is separated from the next by two invisible characters: a carriage return and a line-feed, represented by <cr><lf>. You must compensate for their presence. For example, if the CP is pointing to the beginning of the line, the command 30C moves the CP to the next line:

```
Emily Dickinson said,<cr><lf>
"I fin^d ecstasy in living -<cr><lf>
```

The L (Line) Command

The L command moves the CP the specified number of lines. After an L command, the CP always points to the beginning of a line. The L command takes the following forms:

nL, -nL

where n is the number of lines the CP is to be moved. A positive number moves the CP towards the end of the buffer. A negative number moves the CP back toward the beginning of the buffer. The command 2L moves the CP two lines forward through the memory buffer and positions the character pointer at the beginning of the line.

```
"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
^is joy enough."<cr><lf>
```

The command -L moves the CP to the beginning of the previous line, even if the CP originally points to a character in the middle of the line. Use the special character 0 to move the CP to the beginning of the current line.

The n (Number) Command

The n command moves the CP and displays the destination line. The n command takes the following forms:

n, -n

where n is the number of lines the CP is to be moved. In response to this command, ED moves the CP forward or backward the number of lines specified, then prints only the destination line. For example, the command -2 moves the CP back two lines.

```
Emily Dickinson said,<cr><lf>
^"I find ecstasy in living —<cr><lf>
the mere sense of living<cr><lf>
is joy enough."<cr><lf>
```

A further abbreviation of this command is to enter no number at all. In response to a carriage return without a preceding command, ED assumes a n command of 1 and moves the CP down to the next line and prints it, as follows

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living —<cr><lf>
^the mere sense of living<cr><lf>
```

Also, a minus sign, –, without a number moves the CP back one line.

Displaying Memory Buffer Contents

ED does not display the contents of the memory buffer until you specify which part of the text you want to see. The T command displays text without moving the CP.

The T (Type) Command

The T command types a specified number of lines from the CP at the screen. The T command takes the forms

```
nT, -nT
```

where n specifies the number of lines to be displayed. If a negative number is entered, ED displays n lines before the CP. A positive number displays n lines after the CP. If no number is specified, ED types from the character pointer to the end of the line. The CP remains in its original position no matter how many lines are typed. For example, if the character pointer is at the beginning of the memory buffer, and you instruct ED to type four lines (4T), four lines are displayed at the screen, but the CP stays at the beginning of line 1.

```
^Emily Dickinson said,<cr><lf>
"I find ecstasy in living –<cr><lf>
the mere sense of living<cr><lf>
is joy enough."<cr><lf>
```

If the CP is between two characters in the middle of the line, a T command with no number specified types only the characters between the CP and the end of the line, but the character pointer stays in the same position, as shown in the following memory buffer example:

```
"I find ec^stasy in living –
```

Whenever ED is displaying text with the T command, you can enter a CTRL-S to stop the display, then press any key when you are ready to continue scrolling. Enter a CTRL-C to abort long type-outs.

Deleting Characters

The D (Delete) Command

The D command deletes a specified number of characters and takes the forms:

nD, -nD

where n is the number of characters to be deleted. If no number is specified, ED deletes the character to the right of the CP. A positive number deletes multiple characters to the right of the CP, towards the bottom of the file. A negative number deletes characters to the left of the CP, towards the top of the file. If the character pointer is positioned in the memory buffer as follows

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living –<cr><lf>
the mere sense of living<cr><lf>
is joy ^enough."<cr><lf>
```

the command 6D deletes the six characters after the CP, and the resulting memory buffer looks like this:

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living –<cr><lf>
the mere sense of living<cr><lf>
is joy ^."<cr><lf>
```

You can also use a D command to delete the <cr><lf> between two lines to join them together. Remember that the <cr> and <lf> are two characters.

The K (Kill) Command

The K command kills or deletes whole lines from the memory buffer and takes the forms:

nK, -nK

where n is the number of lines to be deleted. A positive number kills lines after the CP. A negative number kills lines before the CP. When no number is specified, ED kills the current line. If the character pointer is at the beginning of the second line,

```
Emily Dickinson said,<cr><lf>
^"I find ecstasy in living –<cr><lf>
the mere sense of living<cr><lf>
is joy enough."<cr><lf>
```

then the command -K deletes the previous line and the memory buffer changes:

```
^"I find ecstasy in living –<cr><lf>
the mere sense of living<cr><lf>
is joy enough."<cr><lf>
```

If the CP is in the middle of a line, a K command kills only the characters from the CP to the end of the line and concatenates the characters before the CP with the next line. A -K command deletes all the characters between the beginning of the previous line and the CP. A 0K command deletes the characters on the line up to the CP.

You can use the special # character to delete all the text from the CP to the beginning or end of the buffer. Be careful when using #K because you cannot reclaim lines after they are removed from the memory buffer. space

Inserting Characters into the Memory Buffer

The I (Insert) Command

To insert characters into the memory buffer from the screen, use the I command.

If you enter the command in upper-case, ED automatically converts the string to upper-case. The I command takes the forms:

I Istring^Z

When you type the first command, ED enters insert mode. In this mode, all keystrokes are added directly to the memory buffer. ED enters characters in lines and does not start a new line until you press the enter key.

```
A>ED B:QUOTE.TEX

NEW FILE
     : *i
    1: Emily Dickinson said,
    2: "I find ecstasy in living —
    3: the mere sense of living
    4: is joy enough."
    5: ^Z
     : *
```

Note: to exit from insert mode, you must press CTRL-Z or ESC. When the ED prompt, *, appears on the screen, ED is not in insert mode.

In command mode, you can use CP/M Plus command line-editing control characters. In insert mode, you can use the control characters listed in Table 6-3.

Table 6-3. CP/M Plus Line-editing Controls

Command	*Result*
CTRL-H	Delete the last character typed on the current line.
CTRL-U	Delete the entire line currently being typed.
CTRL-X	Delete the entire line currently being typed. Same as CTRL-U.
Backspace	Remove the last character.

When entering a combination of numbers and letters, you might find it inconvenient to press a caps-lock key if your terminal translates the upper-case of numbers to special characters. ED provides two ways to translate your alphabetic input to upper-case without affecting numbers. The first is to enter the insert command letter in upper-case: I. All alphabetics entered during the course of the capitalized command, either in insert mode or as a string, are translated to upper-case. If you enter the insert command letter in lower-case, all alphabetics are inserted as typed. The second method is to enter a U command before inserting text. Upper-case translation remains in effect until you enter a -U command.

The Istring^Z (Insert String) Command

The second form of the I command does not enter insert mode. It inserts the character string into the memory buffer and returns immediately to the ED prompt. You can use CP/M Plus's line-editing control characters to edit the command string.

To insert a string, first use one of the commands that position the CP. You must move the CP to the place where you want to insert a string. For example, if you want to insert a string at the beginning of the first line, use a B command to move the CP to the beginning of the buffer. With the CP positioned correctly, enter an insert string, as follows

```
iIn 1870, ^Z
```

This inserts the phrase "In 1870," at the beginning of the first line, and returns immediately to the ED prompt. In the memory buffer, the CP appears after the inserted string, as follows

```
In 1870, ^Emily Dickinson said,<cr><lf>
```

Replacing Characters

The S (Substitute) Command

The S command searches the memory buffer for the specified string, but when it finds it, automatically substitutes a new string for the search string. Whenever you enter a command in upper-case, ED automatically converts the string to upper-case. The S command takes the form:

```
nSsearch string^Znew string
```

where n is the number of substitutions to make. If no number is specified, ED searches for the next occurrence of the search string in the memory buffer. For example, the command

```
sEmily Dickinson^ZThe poet
```

searches for the first occurrence of "Emily Dickinson" and substitutes "The poet." In the memory buffer, the CP appears after the substituted phrase, as follows

```
The poet^ said,<cr><lf>
```

If upper-case translation is enabled by a capital S command letter, ED looks for a capitalized search string and inserts a capitalized insert string. Note that if you combine this command with other commands, you must terminate the new string with a CTRL-Z.

Combining ED Commands

It saves keystrokes and editing time to combine the editing and display commands. You can type any number of ED commands on the same line. ED executes the command string only after you press the carriage return key. Use CP/M Plus's line-editing controls to manipulate ED command strings.

When you combine several commands on a line, ED executes them in the same order they are entered, from left to right on the command line. There are four restrictions to combining ED commands:

- The combined-command line must not exceed CP/M Plus's 128 character maximum.
- If the combined-command line contains a character string, the line must not exceed 100 characters.

- Commands to terminate an editing session must not appear in a combined-command line.

- Commands, such as the I, J, R, S, and X commands, that require character strings or filespecs must be either the last command on a line or must be terminated with a CTRL-Z or ESC character, even if no character string or filespec is given.

While the examples in the previous section show the memory buffer and the position of the character pointer, the examples in this section show how the screen looks during an editing session. Remember that the character pointer is imaginary, but you must picture its location because ED's commands display and edit text in relation to the character pointer.

Moving the Character Pointer

To move the CP to the end of a line without calculating the number of characters, combine an L command with a C command, L-2C. This command string accounts for the <cr><lf> sequence at the end of the line.

Change the C command in this command string to move the CP more characters to the left. You can use this command string if you must make a change at the end of the line and you do not want to calculate the number of characters before the change, as in the following example:

```
    1: *T
    1: Emily Dickinson said,
    1: *L-7CT
said,
    1: *
```

Displaying Text

A T command types from the CP to the end of the line. To see the entire line, you can combine an L command and a T command. Type 0lt to move the CP from the middle to the beginning of the line and then display the entire line. In the following example, the CP is in the middle of the line. 0L moves the CP to the beginning of the line. T types from the CP to the end of the line, allowing you to see the entire line.

```
    3:  *T
sense of living
    3:  *0LT
    3:  the mere sense of living
    3:  *
```

The command 0TT displays the entire line without moving the CP.

To verify that an ED command moves the CP correctly, combine the command with the T command to display the line. The following example combines a C command and a T command.

```
    2:  *8CT
ecstasy in living –
    2:  *

    4:  *B#T
    1:  Emily Dickinson said,
    2:  "I find ecstasy in living –
    3:  the mere sense of living
    4:  is joy enough."
    1:  *
```

Editing

To edit text and verify corrections quickly, combine the edit commands with other ED commands that move the CP and display text. Command strings like the one that follows move the CP, delete specified characters, and verify changes quickly.

```
1:  *15C5D0LT
1:  Emily Dickinson,
1:  *
```

Combine the edit command K with other ED commands to delete entire lines and verify the correction quickly, as follows

```
1:  *2L2KB#T
```

```
1:  Emily Dickinson said,
2:  "I find ecstasy in living –
1:  *
```

The abbreviated form of the I (insert) command makes simple textual changes. To make and verify these changes, combine the I command string with the C command and the 0LT command string as follows. Remember that the insert string must be terminated by a CTRL-Z.

```
1:  *20Ci to a friend^Z0LT
1:  Emily Dickinson said to a friend,
1:  *
```

Advanced ED Commands

The basic editing commands discussed previously allow you to use ED for all your editing. The following ED commands, however, enhance ED's usefulness.

Moving the CP and Displaying Text

The P (Page) Command

Although you can display any amount of text at the screen with a T command, it is sometimes more convenient to page through the buffer, viewing whole screens of data and moving the CP to the top of each new screen at the same time. To do this, use ED's P command. The P command takes the following forms:

nP, -nP

where n is the number of pages to be displayed. If you do not specify n, ED types the 23 lines following the CP and then moves the CP forward 23 lines. This leaves the CP pointing to the first character on the screen.

To display the current page without moving the CP, enter 0P. The special character 0 prevents the movement of the CP. If you specify a negative number for n, P pages backwards towards the top of the file.

The n: (Line Number) Command

When line numbers are being displayed, ED accepts a line number as a command to specify a destination for the CP. The line number command takes the following form:

```
n:
```

where n is the number of the destination line. This command places the CP at the beginning of the specified line. For example, the command 4: moves the CP to the beginning of the fourth line.

Remember that ED dynamically renumbers text lines in the buffer each time a line is added or deleted. Therefore, the number of the destination line you have in mind can change during editing.

The :n (Through Line Number) Command

The inverse of the line number command specifies that a command should be executed through a certain line number. You can use this command with only three ED commands: the K (kill) command, the L (line) command, and the T (type) command. The :n command takes the following form:

```
:ncommand
```

where n is the line number through which the command is to be executed. The :n part of the command does not move the CP, but the command that follows it might.

You can combine n: with :n to specify a range of lines through which a command should be executed. For example, the command 2::4T types the second, third, and fourth lines:

```
1:  *2::4T
2:  "I find ecstasy in living –
3:  the mere sense of living
4:  is joy enough."
2:  *
```

Finding and Replacing Character Strings

ED supports a find command, F, that searches through the memory buffer and places the CP after the word or phrase you want. The N command allows ED to search through the entire source file instead of just the buffer. The J command searches for and then juxtaposes character strings.

The F (Find) Command

The F command performs the simplest find function; it takes the form:

nFstring

where n is the occurrence of the string to be found. Any number you enter must be positive because ED can only search from the CP to the bottom of the buffer. If you enter no number, ED finds the next occurrence of the string in the file. In the following example, the second occurrence of the word living is found.

```
1:  2fliving
3:  *
```

The character pointer moves to the beginning of the third line where the second occurrence of the word "living" is located. To display the line, combine the find command with a type command. Note that if you follow an F command with another ED command on the same line, you must terminate the string with a CTRL-Z, as follows

```
1:  *2fliving^Z0lt
3:  *the mere sense of living
```

It makes a difference whether you enter the F command in upper- or lower-case. If you enter F, ED internally translates the argument string to upper-case. If you specify f, ED looks for an exact match. For example, Fcp/m plus searches for CP/M Plus but fcp/m plus searches for cp/m plus, and cannot find CP/M Plus.

If ED does not find a match for the string in the memory buffer, it issues the message,

```
BREAK "#" AT
```

where the symbol # indicates that the search failed during the execution of an F command. space

The N Command

The N command extends the search function beyond the memory buffer to include the source file. If the search is successful, it leaves the CP pointing to the first character after the search string. The N command takes the form:

```
nNstring
```

where n is the occurrence of the string to be found. If no number is entered, ED looks for the next occurrence of the string in the file. The case of the N command has the same effect on an N command as it does on an F command. Note that if you follow an N command with another ED command, you must terminate the string with a CTRL-Z.

When an N command is executed, ED searches the memory buffer for the specified string, but if ED does not find the string, it does not issue an error message. Instead, ED automatically writes the searched data from the buffer into the new file. Then ED performs a 0A command to fill the buffer with unsearched data from the source file. ED continues to search the buffer, write out data, and append new data until it either finds the string or reaches the end of the source file. If ED reaches the end of the source file, ED issues the following message:

```
BREAK "#" AT
```

Because ED writes the searched data to the new file before looking for more data in the source file, ED usually writes the contents of the buffer to the new file before finding the end of the source file and issuing the error message.

Note: you must use the H command to continue an edit session after the source file is exhausted and the memory buffer is emptied.

The J (Juxtapose) Command

The J command inserts a string after the search string, then deletes any characters between the end of the inserted string to the beginning of the a third delete-to string. This juxtaposes the string between the search and delete-to strings with the insert string. The J command takes the form:

```
nJsearch string^Zinsert string^Zdelete-to string
```

where n is the occurrence of the search string. If no number is specified, ED searches for the next occurrence of the search string in the memory buffer. In the following example, ED searches for the word "Dickinson", inserts the phrase "told a friend" after it, and then deletes everything up to the comma.

```
1:  *#T 1: Emily Dickinson said,
2:  "I find ecstasy in living –
3:  the mere of living
4:  is joy enough."
1:  *jDickinson^Z told a friend^Z,
1:  *0lt
1:  Emily Dickinson told a friend,
1:  *
```

If you combine this command with other commands, you must terminate the delete-to string with a CTRL-Z or ESC, as in the following example. If an upper-case J command letter is specified, ED looks for upper-case search and delete-to strings and inserts an upper-case insert string.

The J command is especially useful when revising comments in assembly language source code, as follows

```
236:  SORT    LXI    H,  SW     ;ADDRESS   TOGGLE SWITCH
236:  *j;^ZADDRESS SWITCH TOGGLE^Z^L^Z0LT
236:  SORT    LXI    H,  SW     ;ADDRESS   SWITCH TOGGLE
236:  *
```

In this example, ED searches for the first semicolon and inserts ADDRESS SWITCH TOGGLE after the mark and then deletes to the <cr><lf> sequence, represented by CTRL-L. In any search string, you can use CTRL-L to represent a <cr><lf> when the phrase that you want extends across a line break. You can also use a CTRL-I in a search string to represent a tab.

Note: if long strings make your command longer than your screen line length, enter a CTRL-E to cause a physical carriage return at the screen. A CTRL-E returns the cursor to the left edge of the screen, but does not send the command line to ED. Remember that no ED command line containing strings can exceed 100 characters. When you finish your command, press the carriage return key to send the command to ED.

The M (Macro) Command

An ED macro command, M, can increase the usefulness of a string of commands. The M command allows you to group ED commands together for repeated execution. The M command takes the following form:

nMcommand string

where n is the number of times the command string is to be executed. A negative number is not a valid argument for an M command. If no number is specified, the special character # is assumed, and ED executes the command string until it reaches the end of data in the buffer or the end of the source file, depending on the commands specified in the string. In the following example, ED executes the four commands repetitively until it reaches the end of the memory buffer:

```
          1:  *mfliving^Z-6diLiving^Z0lt
          2:  "I find ecstasy in Living –
          3:  the mere sense of Living

    BREAK "#" AT ^Z
3:  *
```

The terminator for an M command is a carriage return; therefore, an M command must be the last command on the line. Also, all character strings that appear in a macro must be terminated by CTRL-Z or ESC. If a character string ends the combined-command string, it must be terminated by CTRL-Z, then followed by a <cr> to end the M command.

The execution of a macro command always ends in a BREAK "#"

message, even when you have limited the number of times the macro is to be performed, and ED does not reach the end of the buffer or source file. Usually the command letter displayed in the message is one of the commands from the string and not M.

To abort a macro command, press a CTRL-C at the keyboard.

The Z (Sleep) Command

Use the Z command to make the editor pause between operations. The pauses give you a chance to review what you have done. The Z command takes the following form:

nZ

where n is the number of seconds to wait before proceeding to the next instruction.

Usually, the Z command has no real effect unless you use it with a macro command. The following example shows you how you can use the Z command to cause a brief pause each time ED finds the word TEXT in a file.

```
A>*mfliving^Z0tt10z
```

Moving Text Blocks

To move a group of lines from one area of your data to another, use an X command to write the text block into a temporary LIB file, then a K command to remove these lines from their original location, and finally an R command to read the block into its new location. space

The X: (Transfer) Command

The X command takes the forms:

nX nXfilespec^Z

where n is the number of lines from the CP towards the bottom of the buffer that are to be transferred to a file. Therefore, n must always be a positive number. The nX command with no file specified creates a temporary file named X$$$$$$$.LIB. This file is erased when you terminate the edit session. The nX command with a file specified creates a file of the specified name. If no filetype is specified, LIB is assumed. This file is saved when you terminate the edit session. If the X command is not the last command on the line, the command must be terminated by a CTRL-Z or ESC. In the following example, just one line is transferred to the temporary file:

```
1:  *X
1:  *t
1:  *Emily Dickinson said,
1:  *kt
1:  *"I find ecstasy in living –
1:  *
```

If no library file is specified, ED looks for a file named X$$$$$$$.LIB. If the file does not exist, ED creates it. If a previous X command already created the library file, ED appends the specified lines to the end of the existing file.

Use the special character 0 as the n argument in an X command to delete any file from within ED.

The R (Read) Command

The X command transfers the next n lines from the current line to a library file. The R command can retrieve the transferred lines. The R command takes the forms:

R Rfilespec

If no filename is specified, X$$$$$$$ is assumed. If no filetype is specified, LIB is assumed. R inserts the library file in front of the CP; therefore, after the file is added to the memory buffer, the CP points to the same character it did before the read, although the character is on a new line number. If you combine an R command with other commands, you must separate the filename from subsequent command letters with a CTRL-Z as in the following example where ED types the entire file to verify the read.

```
1: *41
 : *R^ZB#T
1: "I find ecstasy in living –
2: the mere sense of living
3: is joy enough."
4: Emily Dickinson said,
1: *
```

Saving or Abandoning Changes: ED Exit

You can save or abandon editing changes with the following three commands.

The H (Head of File) Command

An H command saves the contents of the memory buffer without ending the ED session, but it returns to the head of the file. It saves the current changes and lets you reedit the file without exiting ED. The H command takes the following form:

H

followed by a carriage return.

To execute an H command, ED first finalizes the new file, transferring all lines remaining in the buffer and the source file to the new file. Then ED closes the new file, erases any BAK file that has the same file specification as the original source file, and renames the original source file filename.BAK. ED then renames the new file, which has had the filetype $$$, with the original file specification. Finally, ED opens the newly renamed file as the new source file for a new edit, and opens a new $$$ file. When ED returns the * prompt, the CP is at the beginning of an empty memory buffer.

If you want to send the edited material to a file other than the original file, use the following command:

```
A>ED filespec differentfilespec
```

If you then restart the edit with the H command, ED renames the file differentfilename.$$$ to differentfilename.BAK and creates a new file of different filespec when you finish editing.

The O (Original) Command

An O command abandons changes made since the beginning of the edit and allows you to return to the original source file and begin reediting without ending the ED session. The O command takes the form:

```
O
```

followed by a carriage return. When you enter an O command, ED confirms that you want to abandon your changes by asking

```
O (Y/N)?
```

You must respond with either a Y or an N; if you press any other key, ED repeats the question. When you enter Y, ED erases the temporary file and the contents of the memory buffer. When the * prompt returns, the character pointer is pointing to the beginning of an empty memory buffer, just as it is when you start ED.

The Q (Quit) Command

A Q command abandons changes made since the beginning of the ED session and exits ED. The Q command takes the form:

```
Q
```

followed by a carriage return.

When you enter a Q command, ED verifies that you want to abandon the changes by asking

```
Q (Y/N)?
```

You must respond with either a Y or an N; if you press any other key, ED repeats the question. When you enter Y, ED erases the temporary file, closes the source file, and returns control to CP/M Plus.

Note: you can enter a CTRL-Break or a CTRL-C to return control immediately to CP/M Plus. This does not give ED a chance to close the source or new files, but it prevents ED from deleting any temporary files.

ED Error Messages

ED returns one of two types of error messages: an ED error message if ED cannot execute an edit command, or a CP/M Plus error message if ED cannot read or write to the specified file. An ED error message takes the form:

BREAK "x" AT c

where x is one of the symbols defined in the following table and c is the command letter where the error occurred.

Table 6-4. ED Error Symbols

Symbol	*Meaning*
#	Search failure. ED cannot find the string specified in a F, S, or N command.
?c	Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, O, or Q command is not alone on its command line.
0	No .LIB file. ED did not find the LIB file specified in an R command.
>	Buffer full. ED cannot put anymore characters in the memory buffer, or string specified in an F, N, or S command is too long.

E	Command aborted. A keystroke at the keyboard aborted command execution.
F	File error. Followed by either disk FULL or DIRECTORY FULL.

The following examples show how to recover from common editing error conditions. For example

```
BREAK ">" AT A
```

means that ED filled the memory buffer before completing the execution of an A command. When this occurs, the character pointer is at the end of the buffer and no editing is possible. Use the 0W command to write out half the buffer or use an O or H command and reedit the file.

```
BREAK "#" AT F
```

means that ED reached the end of the memory buffer without matching the string in an F command. At this point, the character pointer is at the end of the buffer. Move the CP with a B or n: line number command to resume editing.

```
BREAK "F" AT F DISK FULL
```

Use the 0X command to erase an unnecessary file on the disk or a B#Xd:buffer.sav command to write the contents of the memory buffer onto another disk.

```
BREAK "F" AT n DIRECTORY FULL
```

Use the same commands described in the previous message to recover from this file error.

The following table defines the disk file error messages ED returns when it cannot read or write a file.

Table 6-5. ED Diskette File Error Messages

Message	*Meaning*
CP/M Error on d: BDOS Function = NN	Read/Only File File = FILENAME.TYP Disk d: has Read-Only attribute. This occurs if a different disk has been inserted in the drive since the last cold or warm boot.
** FILE IS READ ONLY **	The file specified in the command to invoke ED has the R/O attribute. ED can read the file so that the user can examine it, but ED cannot change a Read-Only file.

Part 2

PROGRAMMING WITH CP/M PLUS

Section 7
Introduction

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

Memory Organization

PCW8256

The PCW8256's memory is organised into three banks (see Figure 7.1 overleaf):

Bank O is the BDOS bank which contains the banked proportions of the BIOS (Basic Input Output Systems) and BDOS (Basic Disk Operating System) it also contains the screen memory, some disc buffers and the extended BIOS jumpblock (see Appendix J).

Bank 1 is the TPA bank in which all application programs are run.

Bank 2 contains a copy of the CCP (Console Command Processor), disk hash tables, data buffers and parts of the BIOS.

The top 16K of memory is common to all banks, it contains the resident portions of the BIOS and BDOS and part of the TPA.

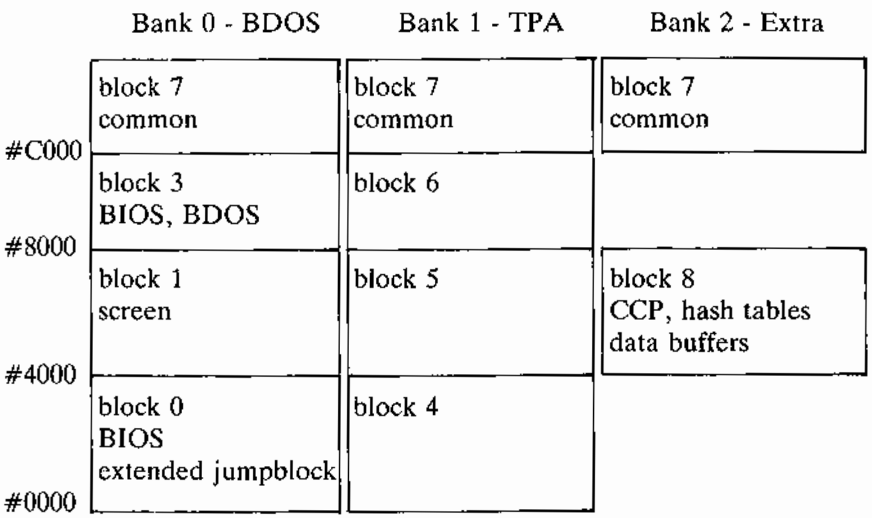


Figure 7-1 PCW8256 Memory Organisation

The screen environment is not included in the above CP/M Plus bank model.

The screen environment consists of blocks 7, 2, 1, 0, i.e. similar to bank 0 but with block 3 replaced by block 2 which contains the matrix RAM, roller RAM and some of the screen RAM. The screen environment can be accessed via the "SCR RUN ROUTINE" entry in the extended BIOS jumpblock (see Appendix J).

The character matrix RAM is at #B800 in the screen environment.

The screen roller RAM is at #B600 in the screen environment.

The RAM disk is in blocks 9..15 plus any additional contiguous memory.

CPC6128

The CPC6128's memory is organised into three banks (Figure 7.2):

Bank 0 is the BDOS bank which contains the banked portions of the BIOS (Basic Input Output System) and BDOS (Basic Disk Operating System) it also contains the screen memory, some disk buffers and the extended BIOS jumpblock (see Appendix J).

Bank 1 is the TPA bank in which all application programs are run.

Bank 2 contains a copy of the CCP (Console Command Processor), disk hash tables and data buffers.

The top 16K of memory is common to all banks, it contains the resident portions of the BIOS and BDOS and part of the TPA.

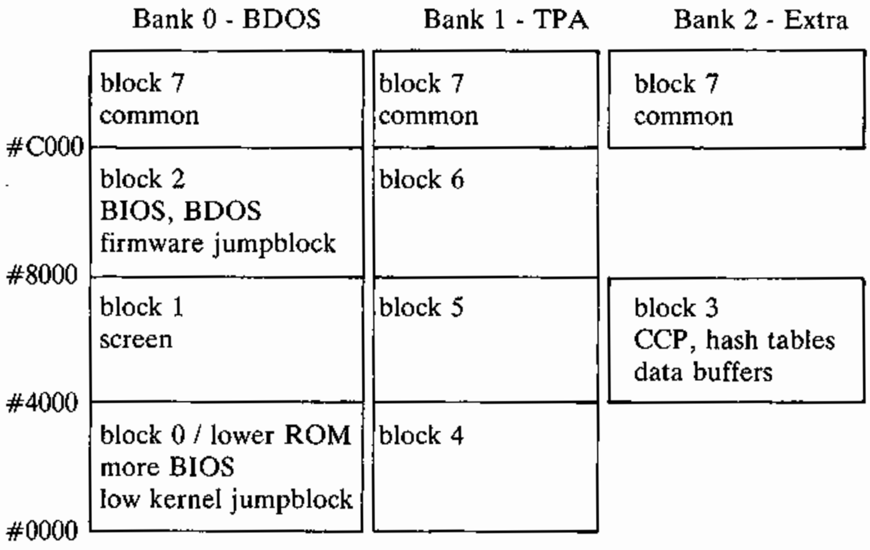


Figure 7.2 CPC6128 Memory Organisation

System Components

Functionally, the CP/M Plus operating system is composed of distinct modules. Transient programs can communicate with these modules to request system services. Figure 7.3 shows the regions where these modules reside in logical memory.

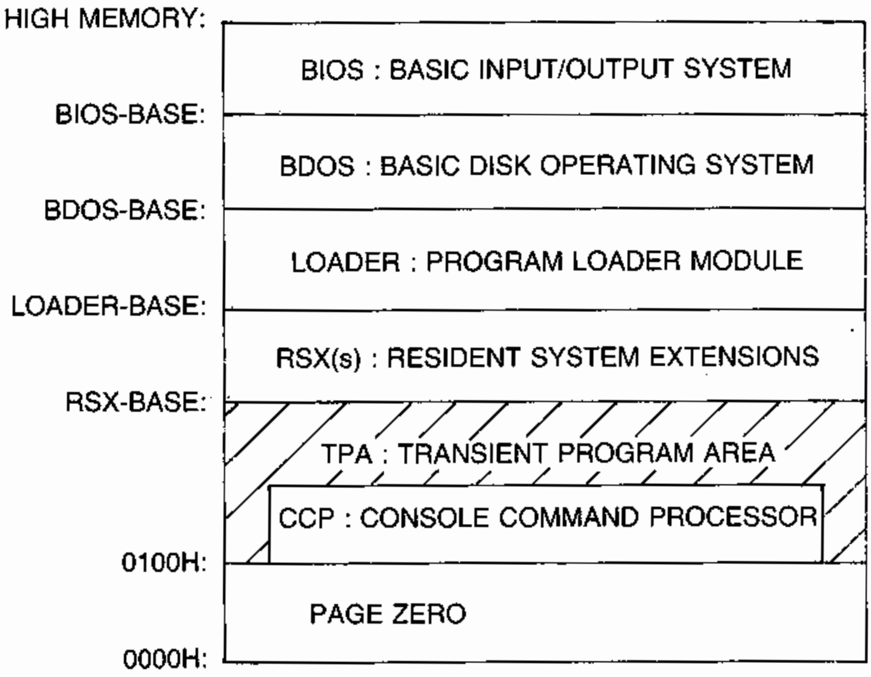

HIGH MEMORY:
BIOS : BASIC INPUT/OUTPUT SYSTEM
BIOS-BASE:
BDOS : BASIC DISK OPERATING SYSTEM
BDOS-BASE:
LOADER : PROGRAM LOADER MODULE
LOADER-BASE:
RSX(s) : RESIDENT SYSTEM EXTENSIONS
RSX-BASE:
TPA : TRANSIENT PROGRAM AREA
CCP : CONSOLE COMMAND PROCESSOR
0100H:
PAGE ZERO
0000H:

Figure 7.3. System Components and Regions in Logical Memory

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

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

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

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

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

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

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

System Component Interaction and Communication

This section describes interaction and communication between the modules and regions defined in "System Components". The most significant channels of communication are between the BDOS and the BIOS, transient programs and the BDOS, and transient programs and RSXs.

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

The BDOS and BIOS

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

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

Applications and the BDOS

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

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

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

Transient programs can terminate execution by jumping to location 0000H in the Page Zero region. This location contains a jump instruction to BIOS-base+3, which contains a jump instruction to the BIOS warm start routine. The BIOS warm start routine loads the CCP into memory at location 100H and then passes control to it.

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

Applications and RSXs

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

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

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

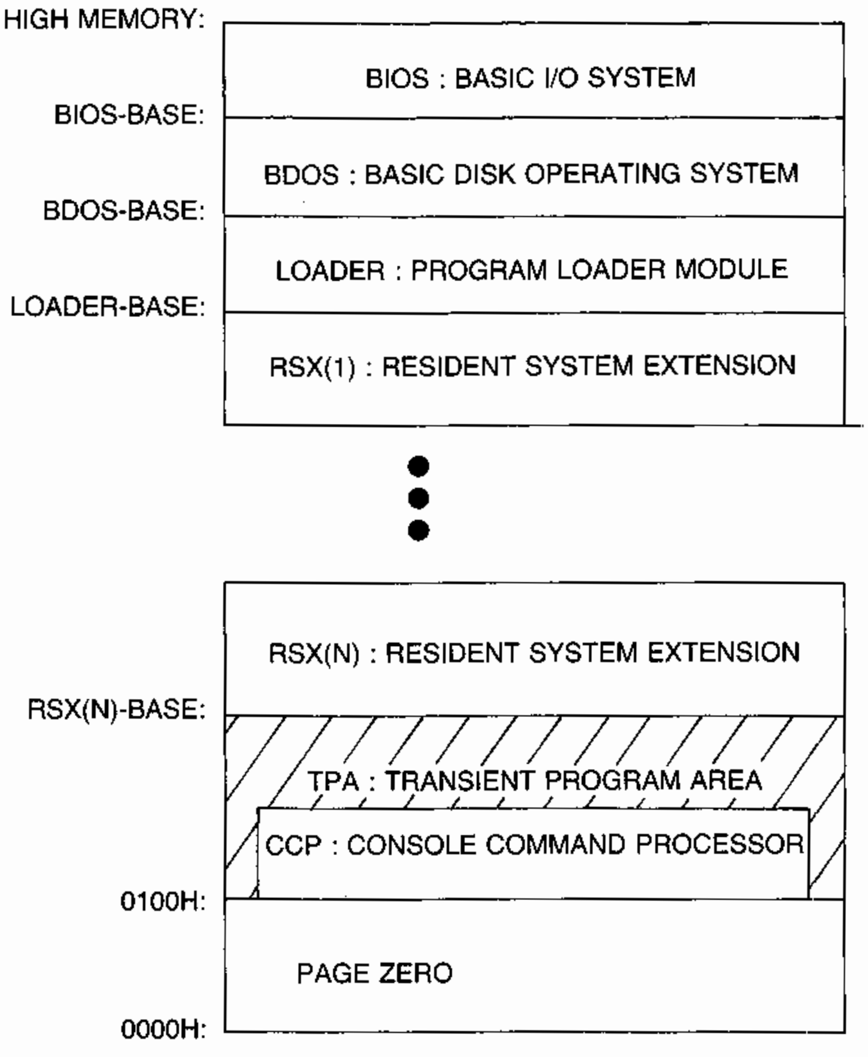

HIGH MEMORY:
BIOS : BASIC I/O SYSTEM
BIOS-BASE:
BDOS : BASIC DISK OPERATING SYSTEM
BDOS-BASE:
LOADER : PROGRAM LOADER MODULE
LOADER-BASE:
RSX(1) : RESIDENT SYSTEM EXTENSION
RSX(N) : RESIDENT SYSTEM EXTENSION
RSX(N)-BASE:
TPA : TRANSIENT PROGRAM AREA
CCP : CONSOLE COMMAND PROCESSOR
0100H:
PAGE ZERO
0000H:

Figure 7-4. System Modules and Regions in Logical Memory

Memory Region Boundaries

This section reviews memory regions under CP/M Plus, and then describes some details of region boundaries. It then relates the sizes of various modules to the space available for the execution of transient programs. Figure 7-4 reviews the location of regions in logical memory.

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

The term High Memory in Figure 7-4 denotes the high address of a CP/M Plus system. This address may fall below the actual top of memory address if space above the operating system has been allocated for directory hashing or data buffering. The maximum top of address is 64K-1 (OFFFFH).

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

RSXs are page aligned modules that are stacked in memory below LOADER-base in memory. In the configuration shown in Figure 7-4, location 0005H of Page Zero contains a jump to location RSX(N)-base + 6. Thus, the memory ceiling of the TPA region is reduced when RSXs are active.

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

Disk and Drive Organization and Requirements

CP/M Plus can support up to sixteen logical drives, identified by the letters A through P, with up to 512 megabytes of storage each. A logical drive

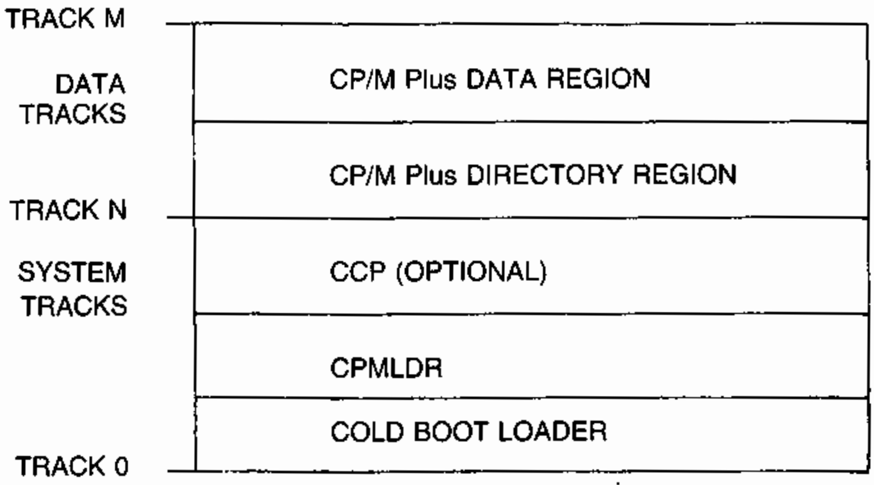


Figure 7-5. Disk Organization

usually corresponds to a physical drive on the system, particularly for physical drives that support removable media such as floppy disks.

In Figure 7-5, the first N tracks are the system tracks. System tracks are required only the disk used by CP/M Plus during system start. All normal CP/M Plus disk access is directed to the data tracks which CP/M Plus uses for file storage.

The data tracks are divided into two regions: a directory area and a data area. The directory area defines the files that exist on the drive and identifies the data space that belongs to each file. The data area contains the file data defined by the directory.

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

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

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

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

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

Disk Organization on PCW8256 and CPC6128

The disc driver supports one or two floppy disc drives. Two disc formats are supported: "system format" and "data format" as per CP/M 2.2 on other AMSTRAD machines.

Only the boot sector is used from the system tracks on a system format disc. The remainder of these tracks are not used.

Many of the disc driver routines are available to an application program via the extended BIOS jumpblock; for example the DISCKIT utility program has to use these routines since there is no standard CP/M Plus facility for formatting discs.

On single drive systems two "logical" drives (A: and B:) are both mapped onto the one physical drive. The number of physical drives fitted is detected during cold boot.

The BIOS routine SELDSK deals with the logical to physical drive mapping. Whenever it is called it checks that the logical drive being selected is currently mapped onto the physical drive. If not a BIOS message is displayed to prompt the user to change the disc.

This enables disc to disc copying using a single drive with standard utilities such as PIP.

The current assignment of logical to physical is displayed on the status line as "Drive is A:" or "Drive is B:".

The disc driver routine interfaces refer to the drives as "unit 0" and "unit 1". This terminology is introduced for reasons of upwards compatability with other products which may have to make a clear distinction between physical drives (units) and logical drives (drives A: B: C: ... P:).

The disc driver routines require "logical" tracks and sectors. These are used to hide information concerning the number of sides and the actual sector numbers from CP/M Plus, which knows nothing about them.

Logical track numbers on a single sided disc are the same as physical track numbers.

For double sided discs two options are available:

Flip sides

side 0 track 0 = logical track	0
side 1 track 0	1

side 0 track 1	2
side 1 track 1	3
...	...

Up and over

side 0 track 0 = logical track	0
side 0 track 1	1
...	...
side 0 last track	
side 1 last track	
side 1 last track − 1	
...	
side 1 track 0	

Logical sectors hide the actual physical sector numbers. Logical sector numbers always start from 0.

Logical sector = physical sector − first sector

BIOS disc error messages are in terms of logical track and sector numbers. All this information is held in an eXtended Disk Parameter Block (see Appendix I).

PCW8256 RAM Disk

Drive M: is a RAM disk. That is, an area of RAM which behaves just like a disk with the exception that all data is lost when the machine is reset or turned off. The size of RAM disk is determined during cold boot. On the standard 256K PCW8256 the RAM disk is 112K. Any contiguous add-on memory is used for the RAM disk.

System Operation

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

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

Cold Start Operation

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

On the CPC6128 the first sector on the system disk (track 0, sector 41) is loaded and given control. This function is performed by bootstrap ROM on PCW8256.

In either case the bootstrap loads the directory and searches for the first file with the extension .EMS. If this file is found then the program file is loaded and given control. The .EMS file contains CPM3.SYS, CCP.COM together with the loader and BIOS.

If the .EMS file is not found then:

- on CPC6128 the message "Cannot find EMS file" is displayed. Press any key to restart.

- on PCW8256 the machine beeps. Press any key to re-execute the bootstrap.

CCP Operation

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

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

When the CCP gains control following a cold start procedure, it displays the system prompt at the console.

```
A>
```

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

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

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

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

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

```
    <command> <command tail>
```

where

```
    <command>       =>  <filespec> or
                        <built-in>

    <command tail>  =>  (no command tail) or
                        <filespec> or <filespec><delimiter><filespec>

    <filespec>      =>  {d:}filename{.typ}{;password}

    <built-in>      =>  one of the CCP built-in commands

    <delimiter>     =>  one or more blanks or a tab or one of the
                        following: "=,[]<>|"

    d:              =>  CP/M Plus drive specification, "A" through "P"

    filename        =>  1 to 8 character filename

    typ             =>  1 to 3 character filetype

    password        =>  1 to 8 character password value
```

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

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

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

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

The following table summarizes the built-in commands.

Table 7-1. CP/M Plus Built-in Commands

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

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

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

```
SUBMIT <command> <command tail>
```

and attempts to load and execute the SUBMIT utility. Thus, the original command field becomes the first command tail field of the SUBMIT command. The procedure the CCP follows to parse a standard command line and execute built-in and transient commands is described as follows:

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

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

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

   Optionally, the CP/M Plus utility SETDEF can specify that a filetype of SUB also be considered when the command filetype field is omitted. When this automatic submit option is in effect, the CCP attempts to open the command with a filetype of COM. If the COM file cannot be found, the CCP repeats the open operation with a filetype of SUB. As an alternative, the order of open operations can be reversed so that the CCP attempts to open with a filetype of SUB first. In either case, the file that is found on disk first determines the filetype field that is ultimately associated with the command.

If the filetype field is present in the command, it must equal COM, SUB or PRL. A PRL file is a Page Relocatable file used in Digital Research's multi-user operating system, MP/M™. Under CP/M Plus, the CCP handles PRL files exactly like COM files.

If the command field is preceded by a drive specification {d:}, the CCP attempts to open the command or submit file on the specified drive. Otherwise, the CCP attempts to open the file on the drives specified in the drive chain.

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

When the current user number is non-zero, all open requests that fail because the file cannot be found, attempt to locate the command file under user zero. If the file exists under user zero with the system attribute set, the file is opened from user zero. This search for a file under user zero is made by the BDOS Open File function. Thus, the user zero open attempt is made before advancing to the next drive in the search chain.

The CCP attempts to open with the first filetype, SUB or COM, on all drives in the search chain before trying the second filetype.

In the banked system, if a password specified in the command field does not match the password of a file on a disk protected in Read mode, the CCP file open operation is terminated with a password error.

If the CCP does not find the command or submit file, it echoes the command line followed by a question mark to the console. If it finds a command file with a filetype of COM or PRL, the CCP proceeds to step 4. If it finds a submit file, it reconstructs the command line as described above, and repeats step 3 for the command, SUBMIT.COM.

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

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

Page Zero initialization is covered in more detail in Section 8.

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

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

If an RSX header is present, the LOADER relocates all RSXs attached to the end of the program, to the top of the TPA region of memory under the LOADER module, or any other RSXs that are already resident. It also updates the address in location 0006H of Page Zero to address the lowest RSX in memory. Finally, the LOADER discards the

RSX header and relocates the program file down one page in memory so that the first executable instruction resides at 100H.

7 After initializing Page Zero, the LOADER module sets up a 32-byte stack with the return address set to location 0000H of Page Zero and jumps to location 100H. At this point, the loaded transient program begins execution.

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

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

Transient Program Operation

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

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

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

Transient programs can use BDOS Function 50, Call BIOS, to access BIOS entry points. This is the preferred method for accessing the BIOS; however, for compatibility with earlier releases of CP/M, transient programs can also make direct BIOS calls for console and list I/O by using the jump instruction at location 0000H in Page Zero. But, to simplify portability, use direct BIOS calls only where the primitive level of functionality provided by the BIOS functions is absolutely required. For example, a disk formatting program must bypass CP/M's disk organization to do its job, and therefore is justified in making direct BIOS calls. Note however, that disk formatting programs are rarely portable.

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

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

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

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

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

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

Resident System Extension Operation

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

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

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

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

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

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

RSX modules are Page Relocatable, PRL, files with the file type RSX. RSX files must be page relocatable because their execution address is determined dynamically by the LOADER module at load time. RSX files have the format shown in Figure 7.6.

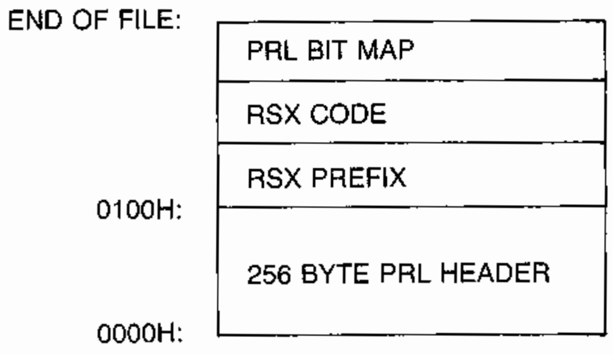


Figure 7-6. RSX File Format

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

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

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

When the CCP gains control after a system warm start, it removes any RSXs in memory that have the Remove flag set to OFFH. All other RSXs remain active in memory. Setting an RSX's Remove flag to 0FFH indicates that the RSX is not active and it can be removed. Note that if an RSX marked for removal is not the lowest active RSX in memory, it still occupies memory after removal. Although the removed RSX cannot be executed, its space is returned to the TPA only when all the lower RSXs are removed.

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

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

SUBMIT Operation

A SUBMIT command line has the following syntax:

SUBMIT <filespec> <parameters>

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

When the SUBMIT utility begins execution, it opens and reads the file specified by <filespec> and creates a temporary submit file of type $$$ on the system's temporary file drive. GENCPM initializes the temporary file drive to the CCP's current default drive. The SETDEF utility can set the temporary file drive to a specific drive. As it creates the temporary file, SUBMIT performs the parameter substitutions requested by the <parameters> subfield of the SUBMIT command line.

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

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

System Control Block

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

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

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

Refer to the description of BDOS Function 49 in Section 8 for more information on the System Control Block. The SCB fields are also discussed in Appendix E.

Section 8
The BDOS System Interface

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

BDOS Calling Conventions

CP/M Plus uses a standard convention for BDOS function calls. On entry to the BDOS, register C contains the BDOS function number, and register pair DE contains a byte or word value or an information address. BDOS functions return single-byte values in register A, and double-byte values in register pair HL. In addition, they return with register A equal to L, and register H equal to B. If a transient program makes a BDOS call to a nonsupported function number in the range of 0 to 127, the BDOS returns with register pair HL set to 0FFFFH. For compatibility with MP/M, the BDOS returns with register pair HL set to 0000H on nonsupported function numbers in the range of 128 to 255. Note that CP/M 2 returns with HL set to zero on all invalid function calls. CP/M Plus's register passing conventions for BDOS function calls are consistent with the conventions used by the Intel PL/M systems programming language.

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

When the CCP loads a transient program, the LOADER module sets the stack pointer to a 16 level stack, and then pushes the address 0000H onto the stack. Thus, an immediate return to the system is equivalent to a jump to 0000H. However, most transient programs set up their own stack, and terminate execution by making a BDOS System Reset call (Function 0) or by jumping to location 0000H.

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

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

BDOS Serial Device I/O

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

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

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

In general, BDOS serial I/O functions read and write an individual ASCII character, or character string to and from these devices, or test the device's ready status. For these BDOS functions, a string of characters is defined as zero to N characters terminated by a delimiter. A block of characters is defined as zero to N characters where N is specified by a word count field.

The maximum value of N in both cases is limited only by available memory. The following list summarizes BDOS serial device I/O functions.

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

CP/M Plus cannot run unless CONIN: and CONOUT: are assigned to a physical console. The remaining logical devices can remain unassigned. If a logical output device is not assigned to a physical device, an output BDOS call to the logical device performs no action. If a logical input device is not assigned to a physical device, an input BDOS call to the logical device typically returns a CTRL-Z (1AH), which indicates end-of-file. Note that these actions depend on your system's BIOS implementation.

BDOS Console I/O

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

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

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

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

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

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

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

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

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

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

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

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

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

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

bit 1 : Setting this bit disables stop and start scroll support for the basic console I/O functions, which comprise the first category of functions described in this section. When this bit is set, Function 1 reads CTRL-S, CTRL-Q, and CTRL-P, and Function 11 returns true if the user types these characters. Use this mode in situations where raw console input and edited output is needed. While in this mode, you can use Function 6 for input and input status, and Functions 1, 9, and 111 for output without the possibility of the output functions intercepting input CTRL-S, CTRL-Q, or CTRL-P characters.

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

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

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

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

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

Other Serial I/O

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

BDOS File System

Transient programs depend on the BDOS file system to create, update, and maintain disk files. This section describes the capabilities of the BDOS file system in detail. You must understand the general features of CP/M Plus described in Section 7 before you can use the detail presented in this section.

The remaining introductory paragraphs define the four categories of BDOS file functions. This is followed by a review of file naming conventions and disk and file organization. The section then describes the data structure used by the BDOS file, and directory oriented functions: the File Control Block (FCB). Subsequent discussions cover file attributes, user numbers, directory labels and extended File Control Blocks (XFCBs), passwords, date and time stamping, blocking and deblocking, multi-sector I/O, disk reset and removable media, byte counts, and error handling. These topics are closely related to the BDOS file system. You must be familiar with the contents of Section 8 before attempting to use the BDOS functions described individually in Section 9.

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

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

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

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

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

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

Disk System Reset
Drive Selection
File Creation
File Open
File Close
Directory Search
File Delete
File Rename
Random or Sequential Read
Random or Sequential Write
Interrogate Selected Disks
Set DMA Address
Set/Reset File Attributes

Reset Drive Set BDOS Multi-Sector Count
Set BDOS Error Mode
Get Disk Free Space
Chain to Program
Flush Buffers
Get/Set System Control Block
Call BIOS
Load Overlay
Call RSX
Truncate File
Set Directory Label
Get File's Date Stamps and Password Mode
Write File XFCB
Set/Get Date and Time
Set Default Password
Return CP/M Plus Serial Number
Get/Set Program Return Code
Parse Filename

File Naming Conventions

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

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

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

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

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

The CCP calls BDOS Function 152, Parse Filename, to parse file specifications from a command line. Function 152 recognizes certain ASCII characters as valid delimiters when it parses a file from a command line. The valid delimiters are shown in Table 8-1. Note that these delimiters are for language 0 only. The LANGUAGE command may alter the character; see your User Guide.

Table 8-1. Valid Filename Delimiters

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

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

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

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

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

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

Disk and File Organization

The BDOS file system can support from one to sixteen logical drives.

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

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

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

Source files under CP/M Plus 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 operation. CTRL-Z characters embedded within machine code files such as COM files are ignored. The actual end-of-file condition returned by the BDOS is used to terminate read operations.

File Control Block Definition

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

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

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

where

dr	drive code (0 - 16) 0 => use default drive for file 1 => auto disk select drive A, 2 => auto disk select drive B, ... 16=> auto disk select drive P.
f1...f8	contain the filename in ASCII upper case, with high bit = 0. f1', ..., f8' denote the high-order bit of these positions, and are file attribute bits.
t1,t2,t3	contain the filetype in ASCII upper case, with high bit = 0. t1', t2', and t3' denote the high bit of these positions, and are file attribute bits. t1' = 1 => Read/Only file t2' = 1 => System file t3' = 1 => File has been archived

ex	contains the current extent number, usually set to 0 by the calling program, but can range 0 - 31 during file I/O
s1	reserved for internal system use
s2	reserved for internal system use
rc	record count for extent "ex" takes on values from 0 - 255 (values greater than 128 imply record count equals 128)
d0...dn	filled-in by CP/M Plus, reserved for system use
cr	current record to read or write in a sequential file operation, normally set to zero by the calling program when a file is opened or created
r0,r1,r2	optional random record number in the range 0-262,143 (0 - 3FFFFH). ro,r1,r2 constitute a 18 bit value with low byte r0, middle byte r1, and high byte r2.

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

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

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

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

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

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

File Attributes

The high-order bits of the FCB filename, f1',...,f8', and filetype, t1',t2',t3', fields are called attribute bits. Attributes bits are 1 bit Boolean fields where 1 indicates on or true, and 0 indicates off or false. Attribute bits indicate two kinds of attributes within the file system: file attributes and interface attributes.

The file attribute bits, f1',...,f4' and t1',t2',t3', can indicate that a file has a defined file attribute. These bits are recorded in a file's directory FCBs. File attributes can be set or reset only by the BDOS Set File Attributes function. When the BDOS Make File function creates a file, it initializes all file attributes to zero. A program can interrogate file attributes in an FCB activated by the BDOS Open File function, or in directory FCBs returned by the BDOS Search For First and Search For Next functions.

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

The file system defines the file attribute bits, t1',t2',t3', as follows:

t1': Read-Only attribute - The file system prevents write operations to a file with the read-only attribute set.

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

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

Attributes f1' through f4' are available for definition by the user.

The interface attributes are indicated by bits f5' through f8' and cannot be used as file attributes. Interface attributes f5' and f6' can request options for BDOS Make File, Close File, Delete File, and Set File Attributes functions. Table 8-2 defines options indicated by the f5' and f6' interface attribute bits for these functions.

Table 8-2. BDOS Interface Attributes

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

Section 9 discusses each interface attribute in detail in the definitions of the above functions. Attributes f5' and f6' are always reset when control is returned to the calling program. Interface attributes f7' and f8' are reserved for internal use by the BDOS file system.

User Number Conventions

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

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

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

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

Directory Labels and XFCBs

The BDOS file system includes two special types of FCBs: the XFCB and the Directory Label. The XFCB is an extended FCB that optionally can be associated with a file in the directory. If present, it contains the file's password.

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

Figure 8-1. XFCB FORMAT

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 reserved area

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

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

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

Figure 8-2. Directory Label Format

dr	-	drive code (0 - 16)
name	-	Directory Label name
type	-	Directory Label type
dl	-	Directory Label data byte
		bit 7 - require passwords for password protected files

	bit 6 - perform access time stamping
	bit 5 - perform update time stamping
	bit 4 - perform create time stamping
	bit 0 - Directory Label exists
	** - bit references are right to left, relative to 0
s1,s2,rc -	n/a
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. The Directory Label name and type fields are not used to search for a Directory Label; they can be used to identify a disk. A Directory Label can be created, or its fields can be updated by BDOS function 100, Set Directory Label. This function can also assign a Directory Label a password. The Directory Label password, if assigned, cannot be circumvented, whereas file password protection is an option controlled by the Directory Label. Thus, access to the Directory Label password provides a kind of super-user status on that drive.

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

File Passwords

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

Password protection is provided in one of three modes. Table 8-3 shows the difference in access level allowed to BDOS functions when the password is not supplied.

If a file is password protected in Read mode, the password must be supplied to open the file. A file protected in Write mode cannot be written

Table 8-3. Password Protection Modes

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

to without the password. A file protected in Delete mode allows read and write access, but the user must specify the password to delete the file, rename the file, or to modify the file's attributes. Thus, password protection in mode 1 implies mode 2 and 3 protection, and mode 2 protection implies mode 3 protection. All three modes require the user to specify the password to delete the file, rename the file, or to modify the file's attributes.

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

Table 8-4 lists the BDOS functions that test for password.

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

Note: The BDOS keeps an assigned default password value until it is replaced with a new assigned value.

Table 8-4. BDOS Functions That Test for Password

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

File Date and Time Stamps

The CP/M 3 File System uses a special type of directory entry called an

The CP/M Plus File System uses a special type of directory entry called an SFCB to record date and time stamps for files. When a directory has been initialized for date and time stamping, SFCBs reside in every fourth position of the directory. Each SFCB maintains the date and time stamps for the previous three directory entries as shown in Figure 8-3.

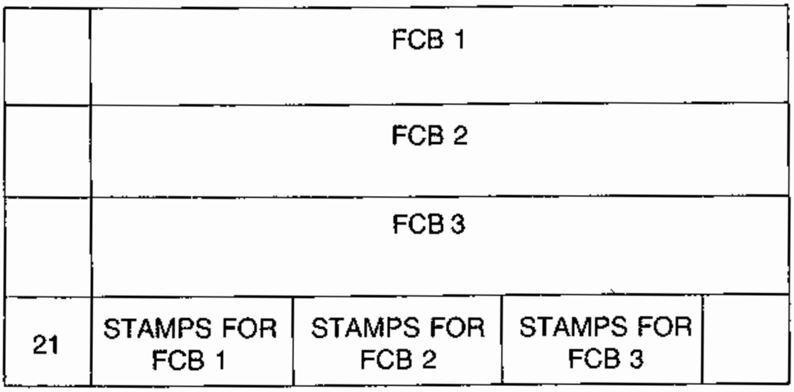


Figure 8-3. Directory Record with SFCB

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

The SFCB directory item contains five fields. The first field is one byte long and contains the value 21H. This value identifies the SFCB in the directory. The next three fields, the SFCB subfields, contain the date and time stamps for their corresponding FCB entries in the directory record. These fields are 10 bytes long. The last byte of the SFCB is reserved for system use. The format of the SFCB subfields is shown in Table 8-5.

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

Table 8-5. SFCB Subfields Format

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

return the file date and time stamps and password mode for a specified file. Refer to Section 9 function 102,. for a description of the format of a date and time stamp field.

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

The CP/M Plus utility, INITDIR, initializes a directory for date and time stamping by placing SFCBs in every fourth directory entry. Date and time stamping is not supported on disks that have not been initialized in this manner. Note that this facility cannot be used if the disk is to be accessed by Locoscript. Locoscript 1.20 does not update access times and earlier versions cannot handle the extended directory. For initialized disks the disks' Directory Label determines the type of date and time stamping supported for files on the drive. If a disk does not have a Directory Label, or if it is Read-Only, or if the disk's Directory Label does not specify date and time stamping, then date and time stamping for files is not performed. Note that the Directory Label is also time stamped, but these stamps are not made in an SFCB. Time stamp fields in the last eight bytes of the Directory Label record when it was created and last updated. Access stamping for Directory Labels is not supported.

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

Record Blocking and Deblocking

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

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

Record deblocking implies a read-ahead operation. For example, if a transient program makes a BDOS function call to read a logical record that resides at the beginning of a physical record, the entire physical record is read into an internal buffer. Subsequent BDOS read calls for the remaining logical records access the buffer instead of the disk. Conversely, record blocking results in the postponement of physical write operations but only for data write operations. For example, if a 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 CP/M Plus, directory write operations are never postponed.

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

Note: the CCP automatically discards all pending physical data buffers when it receives control following a system warm start. However, the

BDOS file system automatically makes a Flush Buffers call in the Close File function. Thus, it is sufficient to close a file to ensure that all pending physical buffers for that file are written to the disk.

Multi-sector I/O

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

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

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

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

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

Disk Reset and Removable Media

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

Logging-in a drive consists of several steps. The most important step is the initialization of the drive's allocation vector. The allocation vector records the allocation and deallocation of data blocks to files, as files are created, extended, deleted, and truncated. Another function performed during drive log-in is the initialization of the directory check-sum vector. The file system uses the check-sum vector to detect media changes on a drive.

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

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

File Byte Counts

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

The BDOS Set File Attributes function can set a file's byte count. Conversely, the Open function can return a file's byte count to the cr field of the FCB. The BDOS Search and Search Next functions also return a file's byte count. These functions return the byte count in the s1 field of the FCB returned in the current DMA buffer (see BDOS Functions Returned 17 and 26).

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

BDOS Error Handling

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

Method 1. It returns to the calling program with return codes in register A, H, and L identifying the error.

Method 2. It displays an error message on the console, and branches to the BIOS warm start entry point, thereby terminating execution of the calling program.

Method 3. It displays an error message on the console, and returns to the calling program as in method 1.

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

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

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

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

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

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

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

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

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

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

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

- Password Error
- File Exists
- ? in Filename

The File Password error is returned when the file password is not supplied, or when it is incorrect.

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

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

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

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

20. Read Sequential
21. Write Sequential

33. Read Random
34. Write Random
40. Write Random w/Zero Fill

The Error Code definitions for register A are shown in Table 8-6.

Table 8-6. Register A BDOS Error Codes

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

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

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

15. Open File
16. Close File
17. Search For First

18. Search For Next
19. Delete File
22. Make File
23. Rename File
30. Set File Attributes
35. Compute File Size
99. Truncate File
100. Set Directory Label
102. Read File Date Stamps and Password Mode
103. Write File XFCB

The Directory Code definitions for register A are shown in Table 8-7.

Table 8-7. BDOS Directory Codes

Code	*Meaning*
00 - 03 :	successful function
255 :	unsuccessful function

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

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

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

The Error Flag definition for register A is shown in Table 8-8.

Table 8-8. BDOS Error Flags

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

The BDOS returns nonzero values in register H to identify a physical or extended error if the BDOS Error Mode is in one of the return modes. Except for functions that return a Directory Code, register A equal to 255 indicates that register H identifies the physical or extended error. For functions that return a Directory Code, if register A equals 255, and register H is not equal to zero, register H identifies the physical or extended error. The physical and extended error codes are shown in Table 8-9.

Table 8-9. BDOS Physical and Extended Errors

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

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

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

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

Page Zero Initialization

Page Zero is the region of memory located from 0000H to 00FFH. This region contains several segments of code and data that are used by transient programs while running under CP/M Plus. The code and data areas are shown in Table 8-10 for reference.

Table 8-10. Page Zero Areas

Location *From* *To*	*Contents*
0000H - 0002H	Contains a jump instruction to the BIOS warm start entry point at BIOS-base + 3. The address at location 0001H can also be used to make direct BIOS calls to the BIOS console status, console input, console output, and list output primitive functions.
0003H - 0004H	(Reserved)
0005H - 0007H	Contains a jump instruction to the BDOS, the LOADER, or to the most recently added RSX, and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, and LHLD 0006H places the address field of the jump instruction in the HL register pair. This value, minus one, is the highest address of memory available to the transient program.
0008H - 003AH	Reserved interrupt locations for Restarts 1 - 7
003BH - 004FH	(Not currently used - reserved)
0050H	Identifies the drive from which the transient

Table 8-10 (continued)

	program was loaded. A value of one to sixteen identifies drives A through P.
0051H - 0052H	Contains the address of the password field of the first command-tail operand in the default DMA buffer beginning at 0080H. The CCP sets this field to zero if no password for the first command-tail operand is specified.
0053H	Contains the length of the password field for the first command-tail operand. The CCP also sets this field to zero if no password for the first command-tail is specified.
0054H - 0055H	Contains the address of the password field of the second command-tail operand in the default DMA buffer beginning at 0080H. The CCP sets this field to zero if no password for the second command-tail operand is specified.
0056H	Contains the length of the password field for the second command-tail operand. The CCP also sets this field .to zero if no password for the second command-tail is specified.
0057H - 005BH	(Not currently used - reserved)
005CH - 007BH	Default File Control Block, FCB, area 1 initialized by the CCP from the first command-tail operand of the command line, if it exists.
006CH - 007BH	Default File Control Block, FCB, area 2 initialized by the CCP from the second command-tail operand of the command line, if it exists.
	Note: this area overlays the last 16 bytes of default FCB area 1. To use the information in this area, a transient program must copy it to another location before using FCB area 1.

007CH	Current record position of default FCB area 1. This field is used with default FCB area 1 in sequential record processing.
007DH - 007FH	Optional default random record position. This field is an extension of default FCB area 1 used in random record processing.
0080H - 00FFH	Default 128-byte disk buffer. This buffer is also filled with the command tail when the CCP loads a transient program.

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

```
    <command> <command tail>
```

where

```
    <command>           =>   <file spec>

<command tail>          =>   (no command tail)
                        =>   <file spec>
                        =>   <file spec><delimiter><file spec>

<file spec>             =>   {d:}filename{.type}{;password}
```

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

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

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

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

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

The Page Zero fields of 0051H through 0056H locate the password fields of the first two file specifications in the command tail if they exist. These fields are provided so that transient programs are not required to parse the command tail for password fields. However, the transient program must save the password, or change the DMA address before performing file operations.

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

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

A hexadecimal dump of 0050H to 00A5H would show the Page Zero initialization performed by the CCP.

```
0050H: 01 8D 00 04 9D 00 08 00 00 00 00 00 02 46 49 4C .............FIL
0060H: 45 20 20 20 20 54 59 50 00 00 00 00 03 46 49 4C E....TYP.....FIL
0070H: 45 20 20 20 20 54 59 50 00 00 00 00 00 00 00 00 E....TYP........
0080H: 24 20 42 3A 46 49 4C 45 2E 54 59 50 3B 50 41 53 . B:FILE.TYP;PAS
0090H: 53 20 43 3A 46 49 4C 45 2E 54 59 50 3B 50 41 53 S C:FILE.TYP;PAS
00A0H: 53 57 4F 52 44 00                                SWORD.
```

Section 9

BDOS Function Calls

This section describes each CP/M Plus system function, including the parameters a program must pass when calling the function, and the values the function returns to the program. The functions are arranged numerically for easy reference. You should be familiar with the BDOS calling conventions and other concepts presented in Section 8 before referencing this section.

BDOS FUNCTION 0: SYSTEM RESET

Entry Parameters:
    Register C: 00H

The System Reset function terminates the calling program and returns control to the CCP via a warm start sequence (see Section 7). Calling this function has the same effect as a jump to location 0000H of Page Zero.

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

BDOS FUNCTION 1: CONSOLE INPUT

Entry Parameters:
    Register C: 01H

Returned Value:
    Register A: ASCII Character

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

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

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

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

BDOS FUNCTION 2: CONSOLE OUTPUT

```
Entry Parameters:
        Register   C: 02H
                   E: ASCII Character
```

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

BDOS FUNCTION 3: AUXILIARY INPUT

```
Entry Parameters:
        Register   C: 03H
```

Returned Value:
    Register A: ASCII Character

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

BDOS FUNCTION 4: AUXILIARY OUTPUT

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

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

BDOS FUNCTION 5: LIST OUTPUT

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

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

BDOS FUNCTION 6: DIRECT CONSOLE I/O

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

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

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

A program calls Function 6 by passing one of four different values in register E. The values and their meanings are summarized in Table 9-1.

Table 9-1. Function 6 Entry Parameters

Register E value	*Meaning*
0FFH	Console input/status command returns an input character; if no character is ready, a value of zero is returned.
0FEH	Console status command (On return, register A contains 00 if no character is ready; otherwise it contains FFH.)
0FDH	Console input command, returns an input character; this function will suspend the calling process until a character is ready.
ASCII character	Function 6 assumes register E contains a valid ASCII character and sends it to the console.

BDOS FUNCTION 7: AUXILIARY INPUT STATUS

Entry Parameters:
    Register C: 07H

Returned Value:
    Register A: Auxiliary Input Status

The Auxiliary Input Status function returns the value 0FFH in register A if a character is ready for input from the logical auxiliary input device, AUXIN:. If no character is ready for input, the value 00H is returned.

BDOS FUNCTION 8: AUXILIARY OUTPUT STATUS

Entry Parameters:
    Register C: 08H

Returned Value:
    Register A: Auxiliary Output Status

The Auxiliary Output Status function returns the value 0FFH in register A if the logical auxiliary output device, AUXOUT:, is ready to accept a character for output. If the device is not ready for output, the value 00H is returned.

BDOS FUNCTION 9: PRINT STRING

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

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

BDOS FUNCTION 10: READ CONSOLE BUFFER

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

Returned Value:
Console Characters in Buffer

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

DE: +0 +1 +2 +3 +4 +5 +6 +7 +8 . . . +n

mx	nc	c1	c2	c3	c4	c5	c6	c7	. . .	??

where mx is the maximum number of characters which the buffer holds, and nc is the number of characters placed in the buffer. The characters entered by the operator follow the nc value. The value mx must be set prior to making a Function 10 call and may range in value from 1 to 255. Setting mx to zero is equivalent to setting mx to one. The value nc is returned to the calling program and may range from zero to mx. If nc < mx, then uninitialized positions follow the last character, denoted by ?? in the figure. Note that a terminating return or line feed character is not placed in the buffer and not included in the count nc.

If register pair DE is set to zero, Function 10 assumes that an initialized input buffer is located at the current DMA address (see Function 26, Set DMA Address). This allows a program to put a string on the screen for the user to edit. To initialize the input buffer, set characters c1 through cn to the initial value followed by a binary zero terminator.

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

Table 9-2. Edit Control Characters

Character	*Edit Control Function*
rub/del	Removes and echoes the last character if at the end of the line; otherwise deletes the character to the left of the current cursor position.
Note: CTRL is given by the ALT key on the PCW8256.	
CTRL-A	Moves cursor one character to the left.
CTRL-B	Moves cursor to the beginning of the line when not at the beginning; otherwise moves cursor to the end of the line.
CTRL-C	Reboots when at the beginning of line; the Console Mode can disable this function.
CTRL-E	Causes physical end of line; if the cursor is positioned in the middle of a line, the characters at and to the right of the cursor are displayed on the next line.
CTRL-F	Moves cursor one character to the right.
CTRL-G	Deletes the character at the current cursor position when in the middle of the line; has no effect when the cursor is at the end of the line.
CTRL-H	Backspaces one character position when at end of line; otherwise deletes character to left of cursor.
CTRL-J	(Line-feed) terminates input; the cursor can be positioned anywhere in the line; the entire input line is accepted; sets the previous line buffer to the input line.
CTRL-K	Deletes all characters to the right of the cursor along with the character at the cursor.
CTRL-M	(RETURN or ENTER have the same effect) terminates input; the cursor can be positioned anywhere in the line; the entire input line is accepted; sets the previous line buffer to the input line.

CTRL-P	Echoes console output to the list device.
CTRL-R	Retypes the characters to the left of the cursor on the new line.
CTRL-U	Updates the previous line buffer to contain the characters to the left of the cursor; deletes current line, and advances to new line.
CTRL-W	Recalls previous line if current line is empty; otherwise moves cursor to end of line.
CTRL-X	Deletes all characters to the left of the cursor.

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

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

BDOS FUNCTION 11: GET CONSOLE STATUS

Entry Parameters:
    Register C: 0BH

Returned Value:
    Register A: Console Status

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

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

BDOS FUNCTION 12: RETURN VERSION NUMBER

Entry Parameters:
Register C: 0CH

Returned Value:
Register HL: Version Number

The Return Version Number function provides information that allows version independent programming. It returns a two-byte value in register pair HL: H contains 00H for CP/M and L contains 31H, the BDOS file system version number. Function 12 is useful for writing applications programs that must run on multiple versions of CP/M and MP/M.

BDOS FUNCTION 13: RESET DISK SYSTEM

Entry Parameters:
Register C: 0DH

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

BDOS FUNCTION 14: SELECT DISK

Entry Parameters:
Register C: 0EH
E: Selected Disk

Returned Value:

Registers	A:	Error Flag
	H:	Physical Error

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

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

Upon return, register A contains a zero if the select operation was successful. If a physical error was encountered, the select function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the select function returns to the calling program with register A set to 0FFH and register H set to one of the following physical error codes:

01 : Disk I/O Error
04 : Invalid drive

BDOS FUNCTION 15: OPEN FILE

Entry Parameters:

Registers	C:	0FH
	DE:	FCB Address

Returned Value:

Registers	A:	Directory Code
	H:	Physical or Extended Error

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

If the file is password protected in Read mode, the correct password must be placed in the first eight bytes of the current DMA, or have been previously established as the default password (see Function 106). If the current record field of the FCB, cr, is set to 0FFH, Function 15 returns the byte count of the last record of the file in the cr field. You can set the last record byte count for a file with Function 30, Set File Attributes. Note that the current record field of the FCB, cr, must be zeroed by the calling program before beginning read or write operations if the file is to be accessed sequentially from the first record.

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

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

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

Upon return, the Open File function returns a directory code in register A with the value 00H if the open was successful, or FFH, 255 decimal, if the file was not found. Register H is set to zero in both of these cases. If a physical or extended error was encountered, the Open File function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, the Open File function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical or extended error codes:

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

BDOS FUNCTION 16: CLOSE FILE

Entry Parameters:

Register	C:	10H
	DE:	FCB Address

Returned Value:

Register	A:	Directory Code
	H:	Physical or Extended Error

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

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

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

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

Upon return, the close function returns a directory code in register A with the value 00H if the close was successful, or FFH, 255 Decimal, if the file was not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the close function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the close function returns to the calling program with register A set to 0FFH and register H set to one of the following physical error codes:

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

BDOS FUNCTION 17: SEARCH FOR FIRST

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

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

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

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

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

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

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

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

If a physical error is encountered, the Search function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the Search function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

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

BDOS FUNCTION 18: SEARCH FOR NEXT

Entry Parameters:

Register	C:	12H

Returned Value:

Register	A:	Directory Code
	H:	Physical Error

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

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

BDOS FUNCTION 19: DELETE FILE

Entry Parameters:

Register	C:	13H
	DE:	FCB Address

Returned Value:

Register	A:	Directory Code
	H:	Extended or Physical Error

The Delete File function removes files or XFCBs that match the FCB addressed in register pair DE. The filename and filetype can contain ambiguous references, that is, question marks in bytes f1 through t3, but the dr byte cannot be ambiguous, as it can in the Search and Search Next functions. Interface attribute f5' specifies the type of delete operation that is performed.

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

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

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

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

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

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

BDOS FUNCTION 20: READ SEQUENTIAL

Entry Parameters:		
Register	C:	14H
	DE:	FCB Address
Returned Value:		
Register	A:	Error Code
	H:	Physical Error

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

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

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

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

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

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

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

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

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

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

BDOS FUNCTION 21: WRITE SEQUENTIAL

Entry Parameters:

Register	C:	15H
	DE:	FCB Address

Returned Value:

Register	A:	Error Code
	H:	Physical Error

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

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

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

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

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

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

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

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

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

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

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

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

BDOS FUNCTION 22: MAKE FILE

Entry Parameters:

Register	C:	16H
	DE:	FCB Address

Returned Value:

Register	A:	Directory Code
	H:	Physical or Extended Error

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

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

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

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

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

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

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

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

Upon return, the Make function returns a directory code in register A with the value 0 if the make operation is successful, or 0FFH, 255 decimal, if no directory space is available. Register H is set to zero in both of these cases. If a physical or extended error is encountered, the Make function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the Make function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical or extended error codes:

01 : Disk I/O error

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

BDOS FUNCTION 23: RENAME FILE

Entry Parameters:

Register	C:	17H
	DE:	FCB Address

Returned Value:

Register	A:	Directory Code
	H:	Physical or Extended Error

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

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

01 : Disk I/O error
02 : Read-Only disk
03 : Read-Only file

04 : Invalid drive error
07 : File password error
08 : File already exists
09 : ? in filename or filetype field

BDOS FUNCTION 24: RETURN LOGIN VECTOR

Entry Parameters:
    Register C: 18H

Returned Value:
    Registers HL: Login Vector

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

BDOS FUNCTION 25: RETURN CURRENT DISK

Entry Parameters:
    Register C: 19H

Returned Value:
    Register A: Current Disk

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

BDOS FUNCTION 26: SET DMA ADDRESS

Entry Parameters:

Register	C:	1AH
	DE:	DMA Address

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

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

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

BDOS FUNCTION 27: GET ADDR(ALLOC)

Entry Parameters:

Register	C:	1BH

Returned Value:

Registers	HL:	ALLOC Address

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

Function 27 returns in register pair HL, the base address of the allocation vector for the currently selected drive. If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), Function 27 returns the value 0FFFFH in the register pair HL.

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

BDOS FUNCTION 28: WRITE PROTECT DISK

Entry Parameters:
    Register C: 1CH

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

BDOS FUNCTION 29: GET READ-ONLY VECTOR

Entry Parameters:
    Register C: 1DH

Returned Value:
    Registers HL: R/O Vector Value

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

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

BDOS FUNCTION 30: SET FILE ATTRIBUTES

Entry Parameters:

Register	C:	1EH
	DE:	FCB Address

Returned Value:

Register	A:	Directory Code
	H:	Physical or Extended error

By calling the Set File Attributes function, a program can modify a file's attributes and set its last record byte count. Other BDOS functions can be called to interrogate these file parameters, but only Function 30 can change them. The file attributes that can be set or reset by Function 30 are f1' through f4', Read-Only, t1', System, t2', and Archive, t3'. The register pair DE addresses an FCB containing a filename with the appropriate attributes set or reset. The calling program must ensure that it does not specify an ambiguous filename. In addition, if the specified file is password protected, the correct password must be placed in the first eight bytes of the current DMA buffer or have been previously established as the default password (see Function 106).

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

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

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

Function 30 searches the referenced directory for entries belonging to the current user number that matches the FCB specified name and type fields.

The function then updates the directory to contain the selected indicators, and if interface attribute f6' is set, the specified byte count value. Note that the last record byte count is maintained in byte 13 of a file's directory FCBs.

File attributes t1', t2', and t3' are defined by CP/M Plus. (They are described in Section 8.3.4.) Attributes f1' through f4' are not presently used, but can be useful for application programs, because they are not involved in the matching program used by the BDOS during Open File and Close File operations. Indicators f5' through f8' are reserved for use as interface attributes.

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

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

BDOS FUNCTION 31: GET ADDR(DPB PARMS)

Entry Parameters:
    Register C: 1FH

Returned Value:
    Registers HL: DPB Address

Function 31 returns in register pair HL the address of the BIOS-resident Disk Parameter Block, DPB, for the currently selected drive. The calling program can use this address to extract the disk parameter values.

If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), Function 31 returns the value 0FFFFH in the register pair HL. See Appendix I for futher details.

BDOS FUNCTION 32: SET/GET USER CODE

Entry Parameters:

Register	C:	20H
	E:	0FFH (get) or User Code (set)

Returned Value:

Register	A:	Current Code or (no value)

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

BDOS FUNCTION 33: READ RANDOM

Entry Parameters:

Register	C:	21H
	DE:	FCB Address

Returned Value:

Register	A:	Error Code
	H:	Physical Error

The Read Random function is similar to the Read Sequential function except that the read operation takes place at a particular random record number, selected by the 24-bit value constructed from the three byte, 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 Function 33, the calling program must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. The base extent may or may not contain any allocated data. Function 33 reads the record specified by the random record field into the current DMA address. The function automatically sets the logical extent and current record values, but unlike the Read Sequential function, it does not advance the current record number. Thus, a subsequent Read Random call rereads the same record. After a random read operation, a file can be accessed sequentially, starting from the current randomly accessed position. However, the last randomly accessed record is reread or rewritten when switching from random to sequential mode.

If the BDOS Multi-Sector count is greater than one (see Function 44), the Read Random function reads multiple consecutive records into memory beginning at the current DMA. The r0, r1, and r2 field of the FCB is automatically incremented to read each record. However, the FCBs random record number is restored to the first record's value upon return to the calling program.

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

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

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

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

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

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

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

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

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

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

BDOS FUNCTION 34: WRITE RANDOM

Entry Parameters:

Register	C:	22H
	DE:	FCB Address

Returned Value:

Register	A:	Error Code
	H:	Physical error

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

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

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

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

If the BDOS Multi-Sector count is greater than one (see Function 44), the Write Random function reads multiple consecutive records into memory beginning at the current DMA. The r0, r1, and r2 field of the FCB is automatically incremented to write each record. However, the FCB's random record number is restored to the first record's value when it returns to the calling program. Upon return, the Write Random function sets register A to zero if the write operation is successful. Otherwise, register A contains one of the following error codes:

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

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

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

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

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

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

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

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

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

BDOS FUNCTION 35: COMPUTE FILE SIZE

```
Entry Parameters:
        Register  C:  23H
                 DE:  FCB Address

Returned Value:
        Register  A:  Error Flag
                  H:  Physical or Extended Error

         Random Record Field Set
```

The Compute File Size function determines the virtual file size, which is, in effect, the address of the record immediately following the end of the file. The virtual size of a file corresponds to the physical size if the file is written sequentially. If the file is written in random mode, gaps might exist in the allocation, and the file might contain fewer records than the indicated size.

To compute file size, the calling program passes in register pair DE the address of an FCB in random mode format, bytes r0, r1 and r2 present. Note that the FCB must contain an unambiguous filename and filetype. Function 35 sets the random record field of the FCB to the random record number + 1 of the last record in the file. If the r2 byte is set to 04, then the file contains the maximum record count 262,144.

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

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

Upon return, Function 35 returns a zero in register A if the file specified by the referenced FCB is found, or an 0FFH in register A if the file is not found. Register H is set to zero in both of these cases. If a physical error is encountered, Function 35 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, Function 35 returns to the calling program with register A set to 0FFH, and register H set to one of the following physical errors:

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

BDOS FUNCTION 36: SET RANDOM RECORD

Entry Parameters:

```
        Register  C:  24H
                 DE:  FCB Address
```

Returned Value: Random Record Field Set

The Set Random Record function returns the random record number of the next record to be accessed from a file that has been read or written sequentially to a particular point. This value is returned in the random record field, bytes r0, r1, and r2, of the FCB addressed by the register pair DE. Function 36 can be useful in two ways.

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

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

BDOS FUNCTION 37: RESET DRIVE

Entry Parameters:

Register	C:	25H
	DE:	Drive Vector

Returned Value:

Register	A:	00H

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

BDOS FUNCTION 38: ACCESS DRIVE

Entry Parameters:

Register	C:	26H

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

BDOS FUNCTION 39: FREE DRIVE

Entry Parameters:

Register	C:	27H

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

BDOS FUNCTION 40: WRITE RANDOM WITH ZERO FILL

Entry Parameters:

Register	C:	28H
	DE:	FCB address

Returned Value:

Register	A:	Error Code
	H:	Physical Error

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

BDOS FUNCTION 41: TEST AND WRITE RECORD

Entry Parameters:

Register	C:	29H
	DE:	FCB Address

Returned Value:

Register	A:	Error Code
	H:	Physical Error

The Test and Write function is an MP/M II..function that is not supported under CP/M Plus. If called, Function 41 returns with register A set to 0FFH and register H set to zero.

BDOS FUNCTION 42: LOCK RECORD

Entry Parameters:

Register	C:	2AH
	DE:	FCB Address

Returned Value:

Register	A:	00H

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

BDOS FUNCTION 43: UNLOCK RECORD

Entry Parameters:

Register	C:	2BH
	DE:	FCB Address

Returned Value:

Register	A:	00H

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

BDOS FUNCTION 44: SET MULTI-SECTOR COUNT

Entry Parameters:

Register	C:	2CH
	E:	Number of Sectors

Returned Value:

Register	A:	Return Code

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

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

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

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

BDOS FUNCTION 45: SET BDOS ERROR MODE

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

Returned Value: None

Function 45 sets the BDOS error mode for the calling program to the mode specified in register E. If register E is set to 0FFH, 255 decimal, the error mode is set to Return Error mode. If register E is set to 0FEH, 254 decimal, the error mode is set to Return and Display mode. If register E is set to any other value, the error mode is set to the default mode.

The SET BDOS Error Mode function determines how physical and extended errors are handled for a program. The Error Mode can exist in three modes: the default mode, Return Error mode, and Return and Display Error mode. In the default mode, the BDOS displays a system message at the console that identifies the error and terminates the calling program. In the return modes, the BDOS sets register A to 0FFH, 255 decimal, places an error code that identifies the physical or extended error in register H and returns to the calling program. In Return and Display mode, the BDOS displays the system message before returning to the calling program. No system messages are displayed, however, when the BDOS is in Return Error mode.

BDOS FUNCTION 46: GET DISK FREE SPACE

Entry Parameters:
    Register C: 2EH
             E: Drive

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

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

fs0	fs1	fs2

Disk Free Space Field Format

```
fs0 = low     byte
fs1 = middle  byte
fs2 = high    byte
```

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

Upon return, register A is set to zero if the function is successful. However, if the BDOS Error Mode is one of the return modes (see Function 45), and a physical error is encountered, register A is set to 0FFH, 255 decimal, and register H is set to one of the following values:

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

BDOS FUNCTION 47: CHAIN TO PROGRAM

Entry Parameters:

Register	C:	2FH
	E:	Chain Flag

The Chain To Program function provides a means of chaining from one program to the next without operator intervention. The calling program must place a command line terminated by a null byte, 00H, in the default DMA buffer. If register E is set to 0FFH, the CCP initializes the default drive and user number to the current program values when it passes control to the specified transient program. Otherwise, these parameters are set to the default CCP values. Note that Function 108, Get/Set Program Return Code, can be used to pass a two byte value to the chained program.

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

BDOS FUNCTION 48: FLUSH BUFFERS

Entry Parameters:

Register	C:	30H
	E:	Purge Flag

Returned Value:

Register	A:	Error Flag
	H:	Physical Error

The Flush Buffers function forces the write of any write-pending records contained in internal blocking/deblocking buffers. If register E is set to 0FFH, this function also purges all active data buffers. Programs that provide write with read verify support need to purge internal buffers to ensure that verifying reads actually access the disk instead of returning data that is resident in internal data buffers. The CP/M Plus PIP utility is an example of such a program.

Upon return, register A is set to zero if the flush operation is successful. If a physical error is encountered, the Flush Buffers function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the Flush Buffers function returns to the calling program with register A set to 0FFH and register H set to the following physical error code:

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

BDOS FUNCTION 49: GET / SET SYSTEM CONTROL BLOCK

```
Entry Parameters:
      Register  C:  31H
               DE:  SCB PB Address

Returned Value:
      Register  A:  Returned Byte
               HL:  Returned Word
```

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

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

```
SCBPB:  DB OFFSET    ; Offset within SCB
        DB SET       ; 0FFH if setting a byte
                     ; 0FEH if setting a word
                     ; 001H - 0FDH are reserved
                     ; 000H if a get operation
        DW VALUE     ; Byte or word value to be set
```

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

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

Table 9-3. System Control Block

Offset	*Description*
00 - 04	Reserved For System Use
05	BDOS version number
06 - 09	User Flags
0A - 0F	Reserved For System Use
10 - 11	Program Error return code
12 - 19	Reserved For System Use
1A	Console Width (columns)
1B	Console Column Position
1C	Console Page Length
1D - 21	Reserved For System Use
22 - 23	CONIN Redirection flag
24 - 25	CONOUT Redirection flag
26 - 27	AUXIN Redirection flag
28 - 29	AUXOUT Redirection flag
2A - 2B	LSTOUT Redirection flag
2C	Page Mode
2D	Reserved For System Use
2E	CTRL-H Active
2F	Rubout Active
30 - 32	Reserved For System Use
33 - 34	Console Mode
35 - 36	Reserved For System Use
37	Output Delimiter
38	List Output Flag
39 - 3B	Reserved For System Use
3C - 3D	Current DMA Address
3E	Current Disk

Table 9-3 (continued)

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

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

BDOS FUNCTION 50: DIRECT BIOS CALLS

```
Entry Parameters:
      Register  C:  32H
               DE:  BIOS PB Address

  Returned Value:   BIOS RETURN
```

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

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

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

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

Under CP/M Plus, direct BIOS calls via the BIOS jump vector are only supported for the BIOS Console I/O, List and USERF functions. You must use Function 50 to call any other BIOS functions. In addition, Function 50 intercepts BIOS Function 27 (Select Memory) calls and returns with register A set to zero.

BDOS FUNCTION 59: LOAD OVERLAY

Entry Parameters:

Register	C:	3BH
	DE:	FCB Address

Returned Value:

Register	A:	Error Code
	H:	Physical Error

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

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

The referenced FCB must be successfully opened before Function 59 is called. The load address is specified in the first two random record bytes of the FCB, r0 and r1. The LOADER returns an error if the load address is less than 100H, or if performing the requested load operation would overlay the LOADER, or any other Resident System Extensions that have been previously loaded.

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

Upon return, Function 59 sets register A to zero if the load operation is successful. If the LOADER RSX is not resident in memory because the calling program did not have a RSX header, the BDOS returns with register A set to 0FFH and register H set to zero. If the LOADER detects an invalid load address, or if insufficient memory is available to load the overlay, Function 59 returns with register A set to 0FEH. All other error returns are consistent with the error codes returned by BDOS Function 20, Read Sequential.

BDOS FUNCTION 60: CALL RESIDENT SYSTEM EXTENSION

```
Entry Parameters:
      Register  C:  3CH
               DE:  RSX PB Address

Returned Value:
      Register  A:  Error Code
                H:  Physical Error
```

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

```
RSXPB:  db FUNC          ; RSX Function number
        db NUMPARMS      ; Number of word parameters
        dw PARMETER1     ; Parameter 1
        dw PARMETER2     ; Parameter 2
        . . .
        dw PARMETERn     ; Parameter n
```

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

BDOS FUNCTION 98: FREE BLOCKS

Entry Parameters:

Register	C:	62H

Returned Value:

Register	A:	Error Flag
	H:	Physical Error

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

Upon return, register A is set to zero if Function 98 is successful. If a physical error is encountered, the Free Blocks function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the Free Blocks function returns to the calling program with register A set to 0FFH and register H set to the following physical error code:

04 : Invalid drive error

BDOS FUNCTION 99: TRUNCATE FILE

Entry Parameters:

Register	C:	63H
	DE:	FCB Address

Returned Value:

Register	A:	Directory Code
	H:	Extended or Physical Error

The Truncate File function sets the last record of a file to the random record number contained in the referenced FCB. The calling program passes the address of the FCB in register pair DE, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and bytes 33 through 35, r0, r1, and r2, specifying the last record number of the file. The last record number is a 24 bit value, stored with the least significant byte first, r0, the middle byte next, r1, and the high byte last, r2. This value can range from 0 to 262,143, which corresponds to a maximum value of 3 in byte r2.

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

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

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

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

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

BDOS FUNCTION 100: SET DIRECTORY LABEL

Entry Parameters:

Register	C:	64H
	DE:	FCB Address

Returned Value:

Register	A:	Directory Code
	H:	Physical or Extended Error

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

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

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

Function 100 also requires that the referenced directory contain SFCBs to activate date and time stamping on the drive. If an attempt is made to activate date and time stamping when no SFCBs exist, Function 100 returns an error code of 0FFH in register A and performs no action. The CP/M Plus INITDIR utility initializes a directory for date and time stamping by placing an SFCB record in every fourth entry of the directory.

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

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

BDOS FUNCTION 101: RETURN DIRECTORY LABEL DATA

Entry Parameters:

```
     Register  C:  65H
               E:  Drive
```

Returned Value:

```
     Registers A:  Directory label
                   Data Byte
               H:  Physical Error
```

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

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

Function 101 returns the directory label data byte to the calling program in register A. Register A equal to zero indicates that no directory label exists on the specified drive. If a physical error is encountered by Function 101 when the BDOS Error mode is in one of the return modes (see Function 45), this function returns with register A set to 0FFH, 255 decimal, and register H set to one of the following:

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

BDOS FUNCTION 102: READ FILE DATE STAMPS AND PASSWORD MODE

Entry Parameters:

Register	C:	66H
	DE:	FCB Address

Returned Value:

Register	A:	Directory Code
	H:	Physical Error

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

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

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

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

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

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

Upon return, Function 102 returns a Directory Code in register A with the value zero if the function is successful, or 0FFH, 255 decimal, if the specified file is not found. Register H is set to zero in both of these cases. If a physical or extended error is encountered, Function 102 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, Function 102 returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

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

BDOS FUNCTION 103: WRITE FILE XFCB

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

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

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

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

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

Upon return, Function 103 returns a Directory Code in register A with the value zero if the XFCB create or update is successful, or 0FFH, 255 decimal, if no directory label exists on the specified drive, or the file named in the FCB is not found, or no space exists in the directory to create an XFCB. Function 103 also returns with 0FFH in register A if passwords are not enabled by the referenced directory's label. On nonbanked systems, this function always returns with register A = 0FFH because passwords are not supported. Register H is set to zero in all of these cases. If a physical or extended error is encountered, Function 103 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, Function 103 returns to the calling program with register A set to 0FFH and register H set to one of the following physical or extended error codes:

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

BDOS FUNCTION 104: SET DATE AND TIME

Entry Parameters:

Register	C:	68H
	DE:	DAT Address

Returned Value: none

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

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

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

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

BDOS FUNCTION 105: GET DATE AND TIME

Entry Parameters:

Register	C:	69H
	DE:	DAT Address

Return Value:

Register	A:	seconds
DAT set		

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

BDOS FUNCTION 106: SET DEFAULT PASSWORD

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

Returned Value: none

The Set Default Password function allows a program to specify a password value before a file protected by the password is accessed. When the file system accesses a password-protected file, it checks the current DMA, and the default password for the correct value. If either value matches the file's password, full access to the file is allowed.

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

BDOS FUNCTION 107: RETURN SERIAL NUMBER

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

Returned Value: Serial number field set

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

BDOS FUNCTION 108: GET/SET PROGRAM RETURN CODE

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

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

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

A program can set or interrogate the Program Return Code by calling Function 108. If register pair DE = 0FFFFH, then the current Program Return Code is returned in register pair HL. Otherwise, Function 108 sets the Program Return Code to the value contained in register pair DE. Program Return Codes are defined in Table 9-4.

Table 9-4. Program Return Codes

Code	*Meaning*
0000 - FEFF	Successful return
FF00 - FFFE	Unsuccessful return

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

BDOS FUNCTION 109: GET/SET CONSOLE MODE

Entry Parameters:
    Register C: 6DH
    DE: 0FFFFH (Get) or Console Mode (Set)

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

A program can set or interrogate the Console Mode by calling Function 109. If register pair DE = 0FFFFH, then the current Console Mode is returned in register HL. Otherwise, Function 109 sets the Console Mode to the value contained in register pair DE.

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

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

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

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

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

The CCP initializes the Console Mode to zero when it loads a program unless the program has an RSX that overrides the default value.

BDOS FUNCTION 110: GET/SET OUTPUT DELIMITER

Entry Parameters:

Register	C:	6EH
	DE:	0FFFFH (Get) or
	E:	Output Delimiter (Set)

Returned Value:

Register	A:	Output Delimiter or (no value)

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

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

BDOS FUNCTION 111: PRINT BLOCK

Entry Parameters:

Register	C:	6FH
	DE:	CCB Address

Returned Value: none

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

The CCB format is:

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

BDOS FUNCTION 112: LIST BLOCK

Entry Parameters:

Register	C:	70H
	DE:	CCB Address

Returned Value: none

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

The CCB format is:

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

BDOS FUNCTION 152: PARSE FILENAME

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

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

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

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

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

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

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

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

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

Function 152 recognizes the following characters as 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 Function 152 encounters a non-graphic character in the range 1 through 31 not listed above, it treats the character as an error. The Parse Filename function initializes the specified FCB shown in Table 9-5.

Table 9-5. FCB Format

Location	*Contents*
byte 0	The drive field is set to the specified drive. If the drive is not specified, the default drive code is used. 0=default, 1=A, 2=B.
byte 1-8	The name is set to the specified filename. All letters are converted to upper-case. If the name is not eight characters long, the remaining bytes in the filename field are padded with blanks. If the filename has an asterisk, *, all remaining bytes in the filename field are filled in with question marks, ?. An error occurs if the filename is more than eight bytes long.
byte 9-11	The type is set to the specified filetype. If no filetype is specified, the type field is initialized to blanks. All letters are converted to upper-case. If the type is not three characters

Table 9-5 (continued)

	long, the remaining bytes in the filetype field are padded with blanks. If an asterisk, *, occurs, all remaining bytes are filled in with question marks, ?. An error occurs if the type field is more than three bytes long.
byte 12-15	Filled in with zeros.
byte 16-23	The password field is set to the specified password. If no password is specified, it is initialized to blanks. If the password is less than eight characters long, remaining bytes are padded with blanks. All letters are converted to upper-case. If the password field is more than eight bytes long, an error occurs. Note that a blank in the first position of the password field implies no password was specified.
byte 24-31	Reserved for system use.

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

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

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

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

Section 10

Programming Examples

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

A Sample File-to-file Copy Program

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

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

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

```
                ;       sample file-to-file copy program
                ;
                ;       at the ccp level, the command
                ;
                ;       copy a:x.y b:u.v
                ;
                ;       copies the file named x.y from drive
                ;       a to a file named u.v on drive b.
                ;
0000 =          boot    equ     0000h   ; system reboot
0005 =          bdos    equ     0005h   ; bdos entry point
005c =          fcb1    equ     005ch   ; first file name
005c =          sfcb    equ     fcb1    ; source fcb
006c =          fcb2    equ     006ch   ; second file name
0080 =          dbuff   equ     0080h   ; default buffer
0100 =          tpa     equ     0100h   ; beginning of tpa
                ;
0009 =          printf  equ     9       ; print buffer func#
000f =          openf   equ     15      ; open file func#
0010 =          closef  equ     16      ; close file func#
0013 =          deletef equ     19      ; delete file func#
0014 =          readf   equ     20      ; sequential read
0015 =          writef  equ     21      ; sequential write
0016 =          makef   equ     22      ; make file func#
                ;
0100                    org     tpa     ; beginning of tpa
0100 311b02             lxi     sp,stack; local stack
                ;
                ;       move second file name to dfcb
0103 0e10               mvi LD C,A c,16 ; half an fcb
0105 116c00  LD DE,[FCB2] lxi   d,fcb2  ; source of move
0108 21da01  LD HL,DFCB lxi     h,dfcb  ; destination fcb
010b 1a LD A,(DE) mfcb: ldax    d       ; source fcb
010c 13        INC ~~[illegible]~~ DE inx d ; ready next
010d 77      LD (HL),A  mov     m,a     ; dest fcb
010e 23      INC HL     inx     h       ; ready next
010f 0d      DEC C      dcr     c       ; count 16...0
0110 c20b01  JP NZ,     jnz     mfcb    ; loop 16 times
                ;
                ;       name has been moved, zero cr
0113 af                 xra     a       ; a = 00h
0114 32fa01             sta     dfcbcr  ; current rec = 0
                ;
                ;       source and destination fcbs ready
                ;
0117 115c00             lxi     d,sfcb  ; source file
011a cd6901             call    open    ; error if 255
011d 118701             lxi     d,nofile ; ready message
0120 3c                 inr     a       ; 255 becomes 0
0121 cc6101             cz      finis   ; done if no file
                ;
                ;       source file open, prep destination
0124 11da01             lxi     d,dfcb  ; destination
0127 cd7301             call    delete  ; remove if present
                ;
012a 11da01             lxi     d,dfcb  ; destination
012d cd8201             call    make    ; create the file
0130 119601             lxi     d,nodir ; ready message
0133 3c                 inr     a       ; 255 becomes 0
0134 cc6101             cz      finis   ; done if no dir space
;
```

```
                ;       source file open, dest file open
                ;       copy until end of file on source
                ;
0137 115c00     copy:   lxi     d,sfcb      ; source
013a cd7801             call    read        ; read next record
013d b7                 ora     a           ; end of file?
013e c25101             jnz     eofile      ; skip write if so
                ;
                ;       not end of file, write the record
0141 11da01             lxi     d,dfcb      ; destination
0144 cd7d01             call    write       ; write record
0147 11a901             lxi     d,space     ; ready message
014a b7                 ora     a           ; 00 if write ok
014b c46101             cnz     finis       ; end if so
014e c33701             jmp     copy        ; loop until eof
                ;
                eofile: ; end of file, close destination
0151 11da01             lxi     d,dfcb      ; destination
0154 cd6e01             call    close       ; 255 if error
0157 21bb01             lxi     h,wrprot    ; ready message
015a 3c                 inr     a           ; 255 becomes 00
015b cc6101             cz      finis       ; should not happen
                ;
                ;       copy operation complete, end
015e 11cc01             lxi     d,normal    ; ready message
                ;
                finis:  ; write message given by de, reboot
0161 0e09               mvi     c,printf
0163 cd0500             call    bdos        ; write message
0166 c30000             jmp     boot        ; reboot system
                ;
                ;       system interface subroutines
                ;       (all return directly from bdos)
                ;
0169            0e0f    open:   mvi         c,openf
016b c30500             jmp     bdos
                ;
016e 0e10       close:  mvi     c,closef
0170 c30500             jmp     bdos
                ;
0173 0e13       delete: mvi c,deletef
0175 c30500             jmp     bdos
                ;
0178 0e14               read:   mvi         c,readf
017a c30500             jmp     bdos
                ;
017d 0e15       write:  mvi     c,writef
017f c30500             jmp     bdos
                ;
0182 0e16       make:   mvi     c,makef
0184 c30500             jmp     bdos
                ;
                ;       console messages
0187 6e6f20f    nofile: db      'no source file$'
0196 6e6f209    nodir:  db      'no directory space$'
01a9 6f7574f    space:  db      'out of data space$'
01bb 7772695    wrprot: db      'write protected?$'
01cc 636f700    normal: db      'copy complete$'
                ;
                ;       data areas
01da            dfcb:   ds      33          ; destination fcb
```

```
01fa =          dfcbcr     equ       dfcb+32     ; current record
                ;
01fb                       ds        32          ; 16 level stack
                stack:
021b                       end
```

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

A Sample File Dump Utility

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

```
                DUMP program reads input file and displays hex data
                ;
0100                       org       100h
0005 =          bdos       equ       0005h       ;dos entry point
0001 =          cons       equ       1           ;read console
0002 =          typef      equ       2           ;type function
0009 =          printf     equ       9           ;buffer print entry
000b =          brkf       equ       11          ;break key function (true if char
000f =          openf      equ       15          ;file open
0014 =          readf      equ       20          ;read function
                ;
005c =          fcb        equ       5ch         ;file control block address
0080 =          buff       equ       80h         ;input disk buffer address
                ;
                ;          non graphic characters
000d =          cr         equ       0dh         ;carriage return
000a =          lf         equ       0ah         ;line feed
                ;
                ;          file control block definitions
005c =          fcbdn      equ       fcb+0       ;disk name
005d =          fcbfn      equ       fcb+1       ;file name
0065 =          fcbft      equ       fcb+9       ;disk file type (3 characters)
```

```
0068 =          fcbrl   equ     fcb+12    ;file's current reel number
006b =          fcbrc   equ     fcb+15    ;file's record count (0 to 128)
007c =          fcbcr   equ     fcb+32    ;current (next) record number (0
007d =          fcbln   equ     fcb+33    ;fcb length
                ;
                ;       set up stack
0100 210000             lxi     h,0
0103 39                 dad     sp
                ;       entry stack pointer in hl from the ccp
0104 221502             shld    oldsp
                ;       set sp to local stack area (restored at finis)
0107 315702             lxi     sp,stktop
                ;       read and print successive buffers
010a cdc101             call    setup     ;set up input file
010d feff               cpi     255       ;255 if file not present
010f c21b01             jnz     openok    ;skip if open is ok
                ;
                ;       file not there, give error message and return
0112 11f301             lxi     d,opnmsg
0115 cd9c01             call    err
0118 c35101             jmp     finis     ;to return
                ;
                openok: ;open operation ok, set buffer index to end
011b 3e80               mvi     a,80h
011d 321302             sta     ibp       ;set buffer pointer to 80h
                ;       hl contains next address to print
0120 210000             lxi     h,0       ;start with 0000
                ;
                gloop:
0123 e5                 push    h         ;save line position
0124 cda201             call    gnb
0127 e1                 pop     h         ;recall line position
0128 da5101     jc      finis     ;carry set by gnb if end file
012b 47                 mov     b,a
                ;       print hex values
                ;       check for line fold
012c 7d                 mov     a,l
012d e60f               ani     0fh       ;check low 4 bits
012f c24401             jnz     nonum
                ;       print line number
0132 cd7201             call    crlf
                ;
                ;       check for break key
0135 cd5901             call    break
                ;       accum lsb = 1 if character ready
0138 0f                 rrc               ;into carry
0139 da5101             jc      finis     ;do not print any more
; 013c 7c               mov     a,h
013d cd8f01     call    phex
0140 7d                 mov     a,l
0141 cd8f01             call    phex
                nonum:
0144 23                 inx     h         ;to next line number
0145 3e20               mvi     a,' '
0147 cd6501             call    pchar
014a 78                 mov     a,b
14b cd8f01              call    phex
014e c32301     jmp     gloop
                ;
                finis:
                ;       end of dump
```

```
0151 cd7201                   call        crlf
0154 2a1502                   lhld        oldsp
 0157 f9                      sphl
                ;             stack pointer contains ccp's stack location

0158 c9                       ret                       ;to the ccp
                ;
                ;
                ;             subroutines
                ;
                break:        ;check break key (actually any key will do)
0159 e5d5c5                   push hl push dl push b; environment saved
15c 0e0b        mvi           c,brkf
015e cd0500                   call        bdos
0161 c1d1e1                   pop bl      pop dl      pop h; environment restored
0164 c9                       ret
                ;
                pchar:        ;print a character
0165 e5d5c5     push hl       push dl     push b;     saved
0168 0e02                     mvi         c,typef
016a 5f                       mov         e,a
016b cd0500                   call        bdos
016e c1d1e1                   pop bl pop dl pop h; restored
0171 c9                       ret         ;           crlf:
0172 3e0d                     mvi         a,cr
0174 cd6501                   call        pchar
0177 3e0a                     mvi         a,lf
0179 cd6501     call          pchar
017c c9                       ret
                ;
                ;
                pnib:         ;print nibble in reg a
017d e60f                     ani         0fh         ;low 4 bits
017f fe0a                     cpi         10
0181 d28901                   jnc         p10
                ;             less than or equal to 9
0184 c630                     adi         '0'
0186 c38b01                   jmp         prn
                ;
                ;             greater or equal to 10
0189 c637       p10:          adi         'a' - 10
018b cd6501     prn:          call        pchar
018e c9                       ret
                ;
                phex:         ;print hex char in reg a
018f f5                       push        psw
0190 0f                       rrc
0191 0f                       rrc
0192 0f                       rrc
0193 0f                       rrc
0194 cd7d01                   call        pnib        ;print nibble
0197 f1                       pop         psw
0198 cd7d01                   call        pnib
019b c9         ret
                ;
                err:          ;print error message
                ;             d,e addresses message ending with "$"
019c 0e09                     mvi         c,printf    ;print buffer function
019e cd0500                   call        bdos
01a1 c9                       ret
                ;
                ;             gnb:        ;get next byte
```

```
01a2 3a1302                 lda         ibp
01a5 fe80                   cpi         80h
01a7 c2b301                 jnz         g0
                ;           read another buffer
                ;
                ;
                ;
01aa cdce01                 call        diskr
01ad b7                     ora         a           ;zero value if read ok
01ae cab301                 jz          g0          ;for another byte
                ;           end of data, return with carry set for eof
01b1 37                     stc
01b2 c9                     ret
                ;
                g0:         ;read the byte at buff+reg a
01b3 5f                     mov         e,a         ;ls byte of buffer index
01b4 1600                   mvi         d,0         ;double precision index to de
01b6 3c         inr         a           ;index=index+1
01b7 321302                 sta         ibp         ;back to memory
                ;           pointer is incremented
                ;           save the current file address
01ba 218000                 lxi         h,buff
01bd 19                     dad         d
                ;           absolute character address is in hl
01be 7e                     mov         a,m
                ;           byte is in the accumulator

01bf b7                     ora         a           ;reset carry bit
01c0 c9                     ret
                ;
                setup:      ;set up file

                ;           open the file for input
01c1 af                     xra         a           ;zero to accum
01c2 327c00                 sta         fcbcr       ;clear current record
                ;
01c5 115c00                 lxi         d,fcb
01c8 0e0f                   mvi         c,openf
01ca cd0500                 call        bdos
                ;           255 in accum if open error
01cd c9                     ret
                ;
                diskr:      ;read disk file record
01ce e5d5c5                 push h!     push d!     push b
01d1 115c00                 lxi         d,fcb
01d4 0e14                   mvi         c,readf
01d6 cd0500                 call        bdos
01d9 c1d1e1                 pop b!      pop d!      pop h
01dc c9                     ret
                ;
                ;           fixed message area
01dd 46494c0signon: db                  'file dump version 2.0$'
01f3 0d0a4e0opnmsg: db                  cr,lf,'no input file present on disk$'

                ;variable area
0213            ibp:        ds          2           ;input buffer pointer
0215            oldsp:      ds          2           ;entry sp value from ccp
                ;
                ;           stack area
0217                        ds          64          ;reserve 32 level stack
                stktop:
                ;
0257                        end
```

A Sample Random Access Program

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

```
A>RANDOM X.DAT
```

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

```
next command?
```

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

```
nW      nR      nF      Q
```

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

```
type data:
```

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

```
error, try again
```

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

```
                ;**************************************************************
                ;*                                                            *
                ;* sample random access program for cp/m 3                    *
                ;*                                                            *
                ;**************************************************************
0100                    org     100h    ;base of tpa
                ;
0000 =          reboot  equ     0000h   ;system reboot
0005 =          bdos    equ     0005h   ;bdos entry point
                ;
0001 =          coninp  equ     1       ;console input function
0002 =          conout  equ     2       ;console output function
0009 =          pstring equ     9       ;print string until '$'
000A =          rstring equ     10      ;read console buffer
 000C =         version equ     12      ;return version number
000F =          openf   equ     15      ;file open function
0010 =          closef  equ     16      ;close function
0016 =          makef   equ     22      ;make file function
0021 =          readr   equ     33      ;read random
0022 =          writer  equ     34      ;write random
0028 =          wrtrzf  equ     40      ;write random zero fill
0098 =          parsef  equ     152     ;parse function
                ;
005C =          fcb     equ     005ch   ;default file control block
 007D =         ranrec  equ     fcb+33  ;random record position
007F =          ranovf  equ     fcb+35  ;high order (overflow) byte
0080 =          buff    equ     0080h   ;buffer address
                ;
000D =          cr      equ     0dh     ;carriage return
000A =          lf      equ     0ah     ;line feed
                ;

                ;**************************************************************
                ;*                                                            *
                ;* load SP, set-up file for random access                     *
                ;*                                                            *
                ;**************************************************************
0100 313703             lxi     sp,stack
                ;
                ;       version 3.1?
0103 0E0C               mvi     c,version
0105 CD0500     call    bdos
0108 FE20               cpi     31h     ;version 3.1 or better?
010A D21601             jnc     versok
                ;       bad version, message and go back
 010D 118102            lxi     d,badver
0110 CD3102             call    print
```

```
0113 C30000             jmp     reboot
                ;
                versok:
                ;       correct version for random access
 0116 0E0F              mvi     c,openf     ;open default fcb
0118 3A5D00     rdname: lda     fcb+1
011B FE20               cpi     ' '
011D C22C01             jnz     opfile
0120 11E002             lxi     d,entmsg
0123 CD3102             call    print
0126 CD2002             call    parse
0129 C31801             jmp     rdname
012C 115C00     opfile: lxi     d,fcb
012F CD0500             call    bdos
0132 3C                 inr     a           ;err 255 becomes zero
0133 C24B01             jnz     ready
                ;
                ;       cannot open file, so create it
0136 0E16               mvi     c,makef
0138 115C00             lxi     d,fcb
013B CD0500             call    bdos
013E 3C                 inr     a           ;err 255 becomes zero
013F C24B01             jnz     ready
                ;
                ;       cannot create file, directory full
0142 11A002             lxi     d,nospace
0145 CD3102             call    print
0148 C30000             jmp     reboot      ;back to ccp
                ;

                ;*****************************************************************
                ;*                                                               *
                ;* loop back to "ready" after each command                       *
                ;*                                                               *
                ;*****************************************************************
                ;
                ready:
                ;       file is ready for processing
                ;
014B CD3C02             call    readcom     ;read next command
014E 227D00             shld    ranrec      ;store input record#
0151 217F00             lxi     h,ranovf
0154 71                 mov     m,c         ;set ranrec high byte
0155 FE51               cpi     'Q'         ;quit?
0157 C26901             jnz     notq
                ;
                ;       quit processing, close file
015A E10                mvi     c,closef
015C 115C00             lxi     d,fcb
015F CD0500             call    bdos
0162 3C                 inr     a           ;err 255 becomes 0
0163 CAFF01             jz      error       ;error message, retry
0166 C30000             jmp     reboot      ;back to ccp
                ;
                ;*****************************************************************
                ;*                                                               *
                ;* end of quit command, process write                            *
                ;*                                                               *
                ;*****************************************************************
                notq:
                ;       not the quit command, random write?
```

```
0169 FE57                   cpi         'W'
016B C29C01                 jnz         notw
                  ;
                  ;         this is a random write, fill buffer until cr
016E 11B302                 lxi         d,datmsg
0171 CD3102                 call        print       ;data prompt
0174 0E7F                   mvi         c,127       ;up to 127 characters
0176 218000                 lxi         h,buff      ;destination
                  rloop:    ;read next character to buff

0179 C5                     push        b           ;save counter
017A E5                     push        h           ;next destination
017B CD0802                 call        getchr      ;character to a
017E E1                     pop         h           ;restore counter
017F C1                     pop         b           ;restore next to fill
0180 FE0D                   cpi         cr          ;end of line?
0182 CA8B01

jz
erloop

                  ;         not end, store character
0185 77                     mov         m,a
0186 23                     inx         h           ;next to fill
0187 0D                     dcr         c           ;counter goes down
0188 C27901                 jnz         rloop       ;end of buffer?
                  erloop:
                  ;         end of read loop, store 00
018B 3600                   mvi         m,0
                  ;
                  ;         write the record to selected record number
018D 0E22                   mvi         c,writer
018F 115C00                 lxi         d,fcb
0192 CD0500                 call        bdos
0195 B7                     ora         a           ;error code zero?
0196 C2FF01                 jnz         error       ;message if not
0199 C34B01                 jmp         ready       ;for another record
                  ;
                  ;
                  ;*****************************************************************
                  ;*                                                               *
                  ;* end of write command, process write random zero fill          *
                  ;*                                                               *
                  ;*****************************************************************
                  notw:
                  ;         not the quit command, random write zero fill?
019C FE46                   cpi         'F'
019E C2CF01                 jnz         notf
                  ;
                  ;         this is a random write, fill buffer until cr

01A1 11B302                 lxi         d,datmsg
01A4 CD3102                 call        print       ;data prompt
01A7 0E7F                   mvi         c,127       ;up to 127 characters
01A9 218000                 lxi         h,buff      ;destination
                  rloop1:   ;read next character to buff
01AC C5                     push        b           ;save counter
01AD E5                     push        h           ;next destination
01AE CD0802                 call        getchr      ;character to a
 01B1 E1                    pop         h           ;restore counter
01B2 C1                     pop         b           ;restore next to fill
```

```
01B3 FE0D                  cpi      cr         ;end of line?
01B5 CABE01                jz       erloop1
                  ;        not end, store character
01B8 77                    mov      m,a
01B9 23                    inx      h          ;next to fill
01BA 0D                    dcr      c          ;counter goes down
01BB C2AC01                jnz      rloop1     ;end of buffer?
                  erloop1:
                  ;        end of read loop, store 00
01BE 3600                  mvi      m,0
                  ;
                  ;        write the record to selected record number
01C0 0E28                  mvi      c,wrtrzf
01C2 115C00                lxi      d,fcb
01C5 CD0500                call                bdos
01C8 B7                    ora      a          ;error code zero?
01C9 C2FF01                jnz      error      ;message if not
01CC C34B01                jmp      ready      ;for another record
                  ;
                  ;*****************************************************************
                  ;*                                                               *
                  ;* end of write commands, process read                           *
                  ;*                                                               *
                  ;*****************************************************************
                  ;
                  notf:
                  ;        not a write command, read record?
01CF FE52                  cpi      'R'
01D1 C2FF01                jnz      error      ;skip if not
                  ;
                  ;        read random record
01D4 0E21                  mvi      c,readr
01D6 115C00                lxi      d,fcb
01D9 CD0500                call     bdos
01DC B7                    ora      a          ;return code 00?
01DD C2FF01                jnz      error
                  ;
                  ;        read was successful, write to console
01E0 CD1502                call     crlf       ;new line
01E3 0E80                  mvi      c,128      ;max 128 characters
01E5 218000                lxi      h,buff     ;next to get
                  wloop:
01E8 7E                    mov      a,m        ;next character
01E9 23                    inx      h          ;next to get
01EA E67F                  ani      7fh        ;mask parity
01EC CA4B01                jz       ready      ;for another command if 00
01EF C5                    push     b          ;save counter
01F0 E5                    push     h          ;save next to get
01F1 FE20                  cpi      ' '        ;graphic?
01F3 D40E02                cnc      putchr     ;skip output if not
01F6 E1                    pop      h
01F7 C1                    pop      b
01F8 0D                    dcr      c          ;count=count-1
01F9 C2E801                jnz      wloop
01FC C34B01                jmp      ready
                  ;
                  ;*****************************************************************
                  ;*                                                               *
                  ;* end of read command, all errors end-up here                   *
                  ;*                                                               *
                  ;*****************************************************************
                  ;
                  error:
```

```
01FF 11BF02                lxi     d,errmsg
0202 CD3102                call    print
0205 C34B01                jmp     ready
                ;
                ;*******************************************************************************
                ;*                                                                             *
                ;* utility subroutines for console i/o                                         *
                ;*                                                                             *
                ;*******************************************************************************
                getchr:
                           ;read next console character to a
0208 0E01                  mvi     c,coninp           .
020A CD0500                call    bdos
020D C9                    ret
                ;
                putchr:
                           ;write character from a to console
020E 0E02                  mvi     c,conout
0210 5F                    mov     e,a     ;character to send
0211 CD0500                call    bdos    ;send character
0214 C9                    ret
                ;
                crlf:
                           ;send carriage return line feed
0215 3E0D                  mvi     a,cr    ;carriage return
0217 CD0E02                call    putchr
021A 3E0A                  mvi     a,lf    ;line feed
021C CD0E02                call    putchr
021F C9                    ret
                ;
                parse:
                           ;read and parse filespec

0220 11F102                lxi     d,conbuf
0223 0E0A                  mvi     c,rstring
0225 CD0500                call    bdos
0228 111303                lxi     d,pfncb
022B 0E98                  mvi     c,parsef
022D CD0500                call    bdos
0230 C9                    ret
                ;
                print:
                           ;print the buffer addressed by de until $
0231 D5                    push    d
0232 CD1502                call    crlf
0235 D1                    pop     d       ;new line
0236 0E09                  mvi     c,pstring
0238 CD0500                call    bdos    ;print the string
023B C9                    ret
                ;
                readcom:
                           ;read the next command line to the conbuf
023C 11D102                lxi     d,prompt
023F CD3102                call    print   ;command?
0242 0E0A                  mvi     c,rstring
0244 11F102                lxi     d,conbuf
0247 CD0500                call    bdos    ;read command line
                ;          command line is present, scan it
024A 0E00                  mvi     c,0     ;start with 00
024C 210000                lxi     h,0     ;          0000
024F 11F302                lxi     d,conlin;  command line
0252 1A         readc:     ldax    d       ;next command character
```

```
 0253 13                 inx     d           ;to next command position
0254 B7                  ora     a           ;cannot be end of command
0255 C8                  rz
                ;        not zero, numeric?
0256 D630                sui     '0'
0258 FE0A                cpi     10          ;carry if numeric
025A D27902     jnc      endrd
                         add-in next digit
025D F5                  push    psw
025E 79                  mov     a,c         ;value in ahl
025F 29                  dad     h
0260 8F         adc      a       ;*2
0261 F5                  push    a           ;save value * 2
0262 E5                  push    h
0263 29                  dad     h           ;*4
0264 8F                  adc     a
0265 29                  dad     h           ;*8
0266 8F                  adc     a
0267 C1                  pop     b           ;*2 + *8 = *10
0268 09                  dad     b
0269 C1                  pop     b
026A 88         adc      b
026B C1                  pop     b           ;+digit
026C 48                  mov     c,b
026D 0600                mvi     b,0
026F 09                  dad     b
0270 CE00                aci     0
0272 4F                  mov     c,a
0273 D25202              jnc     readc
0276 C33C02              jmp     readcom
                endrd:
                ;        end of read, restore value in a
0279 C630                adi     '0'         ;command
027B FE61                cpi     'a'         ;translate case?
027D D8                  rc
                ;        lower case, mask lower case bits
027E E65F                ani     101$1111b
0280 C9                  ret                 ;return with value in chl
                ;
                ;*****************************************************************************
                ;*                                                                           *
                ;* string data area for console messages                                     *
                ;*                                                                           *
                ;*****************************************************************************
                ;
                badver:
0281 736F727279          db      'sorry, you need cp/m version 3$'
                nospace:
02A0 6E6F206469          db      'no directory space$'
                datmsg:
02B3 7479706520          db      'type data: $'
                errmsg:
02BF 6572726F72          db      'error, try again.$'
                prompt:
02D1 6E65787420          db      'next command? $'
                entmsg:
02E0 656E746572          db      'enter filename: $'
                ;
                ;*****************************************************************************
                ;*                                                                           *
                ;* fixed and variable data area                                              *
                ;*                                                                           *
                ;*****************************************************************************
                ;
```

```
02F1 21          conbuf:  db       conlen    ;length of console buffer
02F2             consiz:  ds       1         ;resulting size after read
02F3             conlin:  ds       32        ;length 32 buffer
0021 =           conlen   equ      $-consiz
                 ;
                 pfncb:
0313 F302                 dw       conlin
0315 5C00                 dw       fcb
                 ;
0317                      ds       32        ;16 level stack
                 stack:
0337                      end
```

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

```
GETKEY NAMES.DAT   LASTNAME 10 20
```

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

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

```
QUERY NAMES.DAT LASTNAME.KEY
```

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

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

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

Construction of an RSX Program

This section describes the standard prefix of a Resident System Extension (RSX) and illustrates the construction of an RSX with an example. (See Section 7 for a discussion of how RSXs operate under CP/M Plus.) RSX programs are usually written in assembler, but you can use other languages if the interface between the language and the calling conventions of the BDOS are set up properly.

The RSX Prefix

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

```
serial:
        db      0,0,0,0,0,0
start:
        jmp     ftest                   ; start of program
```

```
next:
            db     0c3h                  ; jump instruction to
            dw     0                     ; next module in line
prev:
            dw     0                     ; previous module
remove:
            db     0ffh                  ; remove flag
nonbank:
            db     0                     ; nonbank flag
name:
            db     '12345678'            ; any 8-character name
loader:
            db     0                     ; loader flag
            db     0,0                   ; reserved area
```

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

For compatibility with previous releases of CP/M, the serial: field of the prefix is set to the serial number of the operating system by the LOADER module when the RSX is loaded into memory. Thus, the address in location 6 locates the byte following the serial number of the operating system with or without RSXs in memory.

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

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

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

The remove: field controls whether the RSX is removed from memory by the next call to the LOADER module via BDOS function 59. If the remove: flag is 0FFH, the LOADER removes the RSX from memory. Note that the CCP always calls the LOADER module during a warm start operation. An RSX that remains in memory past warm start because its remove: flag is zero, must set the flag at its termination to ensure its removal from memory at the following warm start.

The nonbank: field controls when the RSX is loaded. If the field is 0FFH, the LOADER only loads the module into memory on nonbanked CP/M Plus systems. Otherwise, the RSX is loaded into memory under both banked and nonbanked versions of CP/M Plus.

The loader: flag identifies the LOADER RSX. When the LOADER module loads an RSX into memory, it sets this prefix flag of the loaded RSX to zero. However, the loader: flag in the LOADER's prefix contains 0FFH. Thus, this flag identifies the last RSX in the chain, which is always the LOADER.

Example of RSX Use

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

```
                        ; CALLVERS program
0005 =                  bdos      qu        5;                    entry point for BDOS
0009 =                  prtstr    equ       9;                    print string function
000C =                  vers      equ       12                    ; get version function
000D =                  cr        equ       0dh                   ; carriage return
000A =                  lf        equ       0ah                   ; line feed

0100                              org       100h
0100 1605               mvi       d,5       ; Perform 5 times
0102 D5                 loop:     push      d                     ; save counter
0103 0E09                         mvi       c,prtstr
0105 111E01                       lxi       d,call$msg            ; print call message
0108 CD0500                       call      bdos
010B 0E0C                         mvi       c,vers
010D CD0500                       call      bdos                  ; try to get version #
                                                                  ; CALLVERS will intercept
0110 7D                           mov       a,l
0111 323401                       sta       curvers
0114 D1                           pop       d
0115 15                           dcr       d                     ; decrement counter
0116 C20201                       jnz       loop
0119 0E00                         mvi       c,0
```

```
011B C30500                 jmp     bdos
                 call$msg:
011E 0D0A2A2A2A             db      cr,lf,'**** CALLVERS **** $'
0134 00          curvers    db      0
0135                        end

                 ; ECHOVERS RSX
0009 =           pstring    equ     9                    ; string print function
000D =           cr         equ     0dh
000A =           lf         equ     0ah
                 ;                  RSX PREFIX STRUCTURE
                 ;
0000 0000000000             db      0,0,0,0,0,0          ; room for serial number
0006 C31B00                 jmp     ftest                ; begin of program
0009 C3          next:      db      0c3H                 ; jump
000A 0000                   dw      0                    ; next module in line
000C 0000        prev:      dw      0                    ; previous module
000E FF          remov:     db      0ffh                 ; remove flag set
000F 00          nonbnk:    db      0
0010 4543484F56             db      'ECHOVERS'
0018 000000                 db      0,0,0
                 ftest:                                  ; is this function 12?
001B 79                     mov     a,c
001C FE0C                   cpi     12
001E CA2400                 jz      begin                ; yes — intercept
0021 C30900                 jmp     next                 ; some other function
                 begin:
0024 210000                 lxi     h,0
0027 39                     dad     sp                   ; save stack
0028 225400                 shld    ret$stack
002B 317600                 lxi     sp,loc$stack

002E 0E09                   mvi     c,pstring
0030 113E00                 lxi     d,test$msg           ; print message
0033 CD0900                 call    next                 ; call BDOS

0036 2A5400                 lhld    ret$stack            ; restore user stack
0039 F9                     sphl
003A 213100                 lxi     h,0031h              ; return version number
003D C9                     ret

                 test$msg:
003E 0D0A2A2A2A             db      cr,lf,'**** ECHOVERS ****$'
                 ret$stack:
0054 0000                   dw      0
0056                        ds      32                   ; 16 level stack
                 loc$stack:
 0076                       end
```

You can prepare the above programs for execution as follows:

1. Assemble the CALLVERS program using MAC as follows:

```
MAC CALLVERS
```

2. Generate a COM file for CALLVERS with HEXCOM:

   ```
   HEXCOM CALLVERS
   ```

3. Assemble the RSX program ECHOVERS using RMAC:

   ```
   RMAC ECHOVERS
   ```

4. Generate a PRL file using the LINK command:

   ```
   LINK ECHOVERS [OP]
   ```

5. Rename the PRL file to an RSX file:

   ```
   RENAME ECHOVERS.RSX=ECHOVERS.PRL
   ```

6. Generate a COM file with an attached RSX using the GENCOM command:

   ```
   GENCOM CALLVERS ECHOVERS
   ```

7. Run the CALLVERS.COM module:

   ```
   CALLVERS
   ```
   The message

   ```
   **** CALLVERS ****
   ```

   followed by the message

   ```
   **** ECHOVERS ****
   ```

   appears on the screen five times if the RSX program works.

Part 3

APPENDICES

Appendix A

CP/M Plus Messages

Messages come from several different sources. CP/M Plus can display error messages when the Basic Disk Operating System (BDOS) returns an error code. CP/M Plus can also display messages when there are errors in command lines. Each utility supplied with CP/M Plus has its own set of messages. The following table lists CP/M Plus messages and utility messages. If you are running an application program you might see messages other than those listed here. Check the application program's documentation for explanations of those messages.

The messages in Table A-1 might be preceded by ERROR:. Some of them might also be preceded or followed by the filespec of the file causing the error condition. Sometimes the input line is flagged with an up arrow ↑ to indicate the character that caused the error. In this case the message Error at the ^ precedes the appropriate error message. Some of the messages are followed by an additional line preceded by INPUT: OPTION: or DRIVE: followed by the applicable error message.

Table A-1. CP/M Plus Messages

Message	*Meaning*
Assign a password to this file.	
	SET. A password mode has been selected for this file but no password has been assigned.
Auxiliary device redirection not implemented.	
	GET and PUT. AUXIN and AUXOUT cannot be redirected to a file.
Bad character re-enter	
	GENCPM. The character entered was not a number.

Table A-1 (continued)

Bad close.

SAVE. An error occurred during the attempt to close the file probably because the file is write-protected.

Bad Logical Device Assignment;

DEVICE. Only the following logical devices are valid: CONIN: CONOUT: AUXIN: AUXOUT: LST:.

BAD PARAMETER

PIP. You entered an illegal parameter in a PIP command. Retype the entry correctly.

Bad password.

RENAME. The password supplied by the user is incorrect.

Baud rate cannot be set for this device.

DEVICE. Only physical devices that have the SOFT-BAUD attribute can have their baud rates changed. To check the attributes of the physical device type DEVICE physical-dev.

Break "x" at c

ED. "x" is one of the symbols described below and c is the command letter being executed when the error occurred.

#	Search failure. ED cannot find the string specified in an F N or S command.
?	Unrecognized command letter c. ED does not recognize the indicated command letter or an E H O or Q command is not alone on its command line.

O	The file specified in an R command cannot be found.
>	Buffer full. ED cannot put any more characters in the memory buffer or the string specified in an F N or S command is too long.
E	Command aborted. A keystroke at the console aborted command execution.
F	Disk or directory full. This error is followed by either the disk or directory full message. Refer to the recovery procedures listed under these messages.

CANNOT CLOSE: Cannot close file. CANNOT CLOSE FILE. CANNOT CLOSE DESTINATION FILE - filespec

GENCOM HEXCOM LIB-80™, LINK-80, MAC, PIP, RMAC, SUBMIT. An output file cannot be closed. This can occur if the disk is removed before the program terminates.

Cannot delete file.

GENCOM. CP/M cannot delete a file. Check to see if the COM file is Read-Only or password-protected.

Cannot have both create and access time stamps.

SET. CP/M Plus supports either create or access time stamps but not both.

Cannot label a drive with a file referenced.

SET.SET does not allow mixing of files and drives.

CANNOT OPEN SOURCE FILE

HEXCOM. The HEX file is not on the specified drive(s).

Cannot redirect from BIOS.

GET PUT. This message is displayed as a warning only if the system has an invalid BIOS.

Table A-1 (continued)

Cannot set both RO and RW.

SET. A file cannot be set to both Read-Only and Read-Write.

Cannot set both SYS and DIR.

SET. A file cannot be set to both SYS and DIR.

CAN'T DELETE TEMP FILE

PIP. A temporary $$$ file already exists which is Read-Only. Use the SET command to change the attribute to Read-Write then erase it.

CHECKSUM ERROR.
Checksum error

HEXCOM PIP. A hex record checksum error was encountered. The hex record that produced the error must be corrected probably by recreating the hex file.

Close error.

XREF. This message is preceded by the filename.XRF. The disk might have been removed before the program terminated.

Close operation failed.

DISCKIT. There was a problem in closing the file at the end of the file copy operation.

Closing file HELP.DAT
Closing file HELP.HLP

HELP. HELP encountered error while processing the HELP.DAT or the HELP.HLP file.

COM file found and NULL option.

GENCOM. The NULL option implies that no COM file is to be loaded just the RSXs.

.COM file required

DIR, ERA, REN, TYPE. Options in the built-in command line require support from a transient COM file that CP/M Plus cannot find on disk.

COMMON ERROR:

LINK-80. An undefined common block has been selected.

CORRECT ERROR TYPE RETURN OR CTRL-Z

PIP. A hex record checksum was encountered during the transfer of a hex file. The hex file with the checksum error should be corrected probably by recreating the hex file.

CP/M Error on d: Disk I/O
BDOS Function = xx File = filespec

CP/M Plus displays the preceding message if the disk is defective or improperly formatted (wrong density).

CP/M Error on d: Invalid Drive
BDOS Function = xx File = filespec

CP/M Plus displays the preceding message when there is no disk in the drive the drive latch is open or the power is off. It also displays the message when the specified drive is not in the system.

CP/M Error on d: Read/Only Disk
BDOS Function = xx File = filespec

CP/M Plus does not allow you to erase rename update or set attributes of a file residing in a Read-Only drive. Use the SET command to set the drive attribute to Read-Write.

Table A-1 (continued)

CP/M Error on d: Read/Only File BDOS Function = xx File = filespec	CP/M Plus does not allow you to erase rename update or set attributes of a file that is Read-Only. Use the SET command to set the file attribute to Read-Write.
Date and Time Stamping Inactive.	DIR. The DATE option was specified but the disk directory has not been initialized with date/time stamping.
DESTINATION IS R/O DELETE (Y/N)?	PIP. The destination file specified in a PIP command already exists and it is Read-Only. If you type Y the destination file is deleted before the file copy is done. If you type N PIP displays the message **NOT DELETED** and aborts the copy operation.
Device Reassignment Not Supported. Enter new assignment or hit RETURN.	DEVICE. A device assignment is invalid.
Directory already re-formatted.	INITDIR. The directory already has date/time stamping.
Directory full DIRECTORY FULL	ED. There is not enough directory space for the file being written to the destination disk. You can use the 0Xfilespec command to erase any unnecessary files on the disk without leaving the editor.

SUBMIT. There is not enough directory space on the temporary file drive to write the temporary file used for processing SUBMIT files. Use the SETDEF command to determine which drive is the temporary file drive. Use the ERASE command to erase unnecessary files or set the temporary file drive to a different drive and retry.

LIB-80 LINK-80. There is no directory space for the output or intermediate files. Use the ERASE command to remove unnecessary files.

HEXCOM. There is no directory space for the output COM file.

Directory needs to be reformatted for date/time stamps.

SET. A date/time option was specified but the directory has not been initialized for date/time stamping. Use the INITDIR command to initialize the directory for date/time stamping.

DISK FULL

ED. There is not enough disk space for the output file. This error can occur on the E H W or X commands. If it occurs with X command you can repeat the command prefixing the filename with a different drive.

DISK READ
DISK READ ERROR:
Disk read error: filespec
DISK READ ERROR - filespec

HEXCOM LIB-80 LINK-80 PIP. The disk file specified cannot be read.

Table A-1 (continued)

DISK WRITE.
Disk Write Error
DISK WRITE ERROR:
DISK WRITE ERROR - filespec

> HEXCOM LIB-80 LINK-80 PIP SUBMIT. A disk write operation cannot be successfully performed probably because the disk is full. Use the ERASE command to remove unnecessary files.

Do you want another file? (Y/N)

> PUT. Enter Y to redirect output to an additional file. Otherwise enter N.

Drive defined twice in search path

> SETDEF. A drive can be specified only once in the search path order.

Drive Read Only

> ERASE RENAME. The specified file is on a Read-Only drive and cannot be erased or renamed.

Duplicate RSX in header. Replacing old by new.
This file was not used.

> GENCOM. The specified RSX is already attached to the COM file. The old one is discarded.

Duplicate input RSX.

> GENCOM. Two or more RSXs of the same name are specified. GENCOM uses only one of the RSXs.

END OF FILE^Z ?

> PIP encountered an unexpected end-of-file during a HEX file transfer.

End of line expected.

DEVICE GET PUT SETDEF. The command typed does not have any further parameters. An end-of-line was expected. Any further characters on the line were ignored.

Error at end of line:

DEVICE GET PUT SETDEF. The error detected occurred at the end of the input line.

Error on line nnnnn:

SUBMIT. The SUBMIT program displays its messages in the preceding format where nnnnn represents the line number of the SUBMIT file. Refer to the message following the line number for explanation of the error.

FILE ERROR

ED. Disk or directory is full and ED cannot write anything more on the disk. This is a fatal error so make sure there is enough space on the disk to hold a second copy of the file before invoking ED.

File already exists; Delete it? (Y/N)
file already exists delete (Y/N)?

PUT. Enter Y to delete the file. Otherwise the program terminates.
RENAME. The above message is preceded by filespec. You have asked CP/M Plus to create or rename a file using a file specification that is already assigned to another file. Either delete the existing file or use another file specification.

File exists erase it

ED. The destination filename already exists when you are placing the destination file on a different disk than the source. It should be erased or another disk selected to receive the output file.

Table A-1 (continued)

FILE IS READ/ONLY
File is Read Only

> ED. The file specified in the command to invoke ED has the Read-Only attribute. ED can read the file so that you can examine it but ED cannot change a Read-Only file. PUT. The file specified to receive the output is a Read-Only file.

FILE NAME ERROR:

> LIB-80. The form of a source filename is invalid.

File not found.
FILE NOT FOUND - filespec

> DUMP ED GENCOM GET PIP SET. An input file that you have specified does not exist. Check that you have entered the correct drive specification or that you have the correct disk in the drive.

First submitted file must be a COM file.

> GENCOM. A COM file is expected as the first file in the command tail. The only time GENCOM does not expect to see a COM file in the first position of the command tail is when the NULL option is specified.

FIRST COMMON NOT LARGEST:

> LINK-80. A subsequent COMMON declaration is larger than the first COMMON declaration for the indicated block. Check that the files being linked are in the proper order or that the modules in a library are in the proper order.

HELP.DAT not on current drive.

> HELP. HELP cannot find HELP.DAT file to process.

Illegal command tail.

DIR. The command line has an invalid format or option.

Illegal Format Value.

DIR. Only SIZE and FULL options can be used for display formats.

Illegal Global/Local Drive Spec Mixing.

DIR. Both a filespec with a drive specifier and the DRIVE option appears in the command.

Illegal filename.

SAVE. There is an error in the filespec on the command line.

Illegal Option or Modifier.

DIR. An invalid option or abbreviation was used.

Illegal date/time specification.

DATE. Date/time format is invalid.

Incorrect file specification.

RENAME. The format of the filespec is invalid.

INDEX ERROR:

LINK-80. The index of an IRL contains invalid information.

Insufficient Memory
INSUFFICIENT MEMORY:

GET LINK-80 PUT SUBMIT. There is not enough memory to allocate buffers or there are too many levels of SUBMIT nesting.

Table A-1 (continued)

Invalid ASCII character

SUBMIT. The SUBMIT file contains an invalid character (0FFH).

Invalid command.

GET and PUT. The string or substring typed in the command line was not recognized as a valid command in the context used.

Invalid delimiter.

DEVICE GET PUT SETDEF. The delimiter [ ] = or space — was not valid at the location used. For example a [was used where an = should have been used.

INVALID DESTINATION:

PIP. An invalid drive or device was specified.

INVALID DIGIT - filespec

PIP. An invalid hex digit has been encountered while reading a hex file. The hex file with the invalid hex digit should be corrected probably by recreating the hex file.

Invalid drive.

SETDEF, TYPE. The specified drive was not a valid drive. Drives recognized by SETDEF and TYPE are A, B and M.

Invalid File.
INVALID FILENAME
Invalid file name.
Invalid Filename.
Invalid file specification.

ED, ERASE, GENCOM, GET, PIP, PUT, SET, SUBMIT, TYPE. The filename typed does not conform to the normal CP/M PLUS file naming conventions.

INVALID FORMAT

PIP. The format of your PIP command is illegal. See the description of the PIP command.

INVALID HEX DIGIT.

HEXCOM. An invalid hex digit has been encountered while reading a hex file. The hex file with the invalid hex digit should be corrected by recreating the hex file.

Invalid number.

DEVICE. A number was expected but not found or the number was out of range; numbers must be from 0 to 255.

Invalid option.

DEVICE and GET. An option was expected and the string found was not a device option or was not valid in the context used.
SETDEF. The option typed in the command line is not a valid option. Valid options are DISPLAY NO DISPLAY NO PAGE ORDER PAGE TEMPORARY.

Invalid option or modifier.

DIR, GET, PUT. The option typed is not a valid option.

INVALID PARAMETER:

MAC, RMAC. An invalid assembly parameter was found in the input line. The assembly parameters are printed at the console up to the point of the error.

Table A-1 (continued)

INVALID PASSWORD
Invalid password or passwords not allowed.

ED PIP. The specified password is incorrect or a password was specified but the file is not password-protected.

Invalid physical device.

DEVICE. A physical device name was expected. The name found in the command string does not correspond to any physical device name in the system.

INVALID REL FILE:

LINK-80. The file indicated contains an invalid bit pattern. Make sure that a REL or IRL file has been specified.

Invalid RSX type.

GENCOM. Filetype must be RSX.

Invalid SCB offset.

GENCOM. The specified SCB is out of range. The SCB offset range is 00H-64H.

INVALID SEPARATOR

PIP. You have placed an invalid character for a separator between two input filenames.

INVALID SOURCE

PIP. An invalid drive or device was specified. AUX and CON are the only valid devices.

Invalid type for ORDER option.

SETDEF. The type specified in the command line was not COM or SUB.

Invalid SYM file format

XREF. The filename.SYM file input to XREF is invalid.

INVALID USER NUMBER

PIP. You have specified a user number greater than 15. User numbers are in the range 0 to 15.

Invalid wildcard.

RENAME. The filespec contained an invalid wildcard specification.

Invalid wild card in the FCB name or type field.

GENCOM. GENCOM does not allow wildcards in filespecs.

LOAD ADDRESS LESS THAN 100.

HEXCOM. The program origin is less than 100H.

MAIN MODULE ERROR:

LINK-80. A second main module was encountered.

Make error

XREF. There is not more directory space on the specified drive.

MEMORY OVERFLOW:

LINK-80. There is not enough memory to complete the link operation.

Missing Delimiter or Unrecognized Option.

ERASE. The ERASE command line format is invalid.

Table A-1 (continued)

Missing left parenthesis.

GENCOM. The SCB option must be enclosed by a left parenthesis.

Missing right parenthesis.

GENCOM. The SCB option is not enclosed with a right parenthesis.

Missing SCB value.

GENCOM. The SCB option requires a value.

More than four drives specified.

SETDEF. More than four drives were specified for the drive search chain.

MULTIPLE DEFINITION:

LINK-80. The specified symbol is defined in more than one of the modules being linked.

n?

USER. You specified a number greater than fifteen for a user area number. For example if you type USER 18 the screen displays 18?.

No directory label exists.

SHOW. The LABEL option was requested but the disk has no label.

No directory space
NO DIRECTORY SPACE - filespec

GENCOM MAC PIP RMAC AND SAVE. There is not enough directory space for the output file. Use the ERASE command to remove unnecessary files on the disk and try again.

No disk space.

SAVE. There is not enough space on the disk for the output file. Use the SHOW command to display the amount of disk space left and use the ERASE command to remove unnecessary files from the disk or use another disk with more file space.

No file
NO FILE:
NO FILE - filespec

DIR ERASE LIB-80 LINK-80 PATCH PIP RENAME TYPE. The specified file cannot be found in the specified drive(s).

No HELP.HLP file on the default drive.

HELP. The file HELP.HLP must be on the default drive.

NO INPUT FILE PRESENT ON DISK

DUMP. The file you requested does not exist.

No memory

There is not enough memory available for loading the specified program.

No modifier for this option.

GENCOM. A modifier was specified but none is required.

NO MODULE:

LIB-80. The indicated module cannot be found.

Table A-1 (continued)

No more space in the header for RSXs or SCB initialization.

GENCOM. The header has room for only 15 entries or the combination of RSXs and SCBs exceeds the maximum.

No options specified.

SET. Specify an option.

No PRN file.

XREF. The file filename.PRN is not present on the specified drive.

No Records Exist

DUMP. Only a directory entry exists for the file.

NO SOURCE FILE PRESENT:

MAC RMAC. The source file cannot be found on the specified drive.

NO SPACE

SAVE. There is no space in the directory for the file being written.

No 'SUB' file found.

SUBMIT. The SUB file typed in the command line cannot be found in the drive search process.

No such file to rename.

RENAME. The file to be renamed does not exist on the specified drive(s).

No SYM file

XREF. The file filename.SYM is not present on the specified drive.

NON-SYSTEM FILE(S) EXIST

DIRS. If nonsystem (DIR) files reside on the specified drive DIRS displays this message.

Not enough available memory. Not Enough Memory Not Enough Memory for Sort.

DIR INITDIR. There is not enough memory for data or sort buffers.

Not enough room in directory.

INITDIR. There is not enough remaining directory space to allow for the date and time extension.

NOT FOUND

PIP. PIP cannot find the specified file.

Not renamed filespec read only.

RENAME. The specified file cannot be renamed because it is Read-Only.

OPEN FILE NONRECOVERABLE

PIP. A disk has the wrong format or a bad sector.

Option only for drives.

SET. The specified option is not valid for files.

Option requires a file reference.

SET. The specified option requires a filespec.

Table A-1 (continued)

Options not grouped together.	DIR. Options can only be specified within one set of brackets.
Output File Exists Erase it.	The output file specified must not already exist.
OUTPUT FILE READ ERROR:	MAC RMAC. An output file cannot be written properly probably because the disk is full. Use the ERASE command to delete unnecessary files from the disk.
OVERLAPPING SEGMENTS:	LINK-80. LINK-80 attempted to write a segment into memory already used by another segment.
Page and nopage option selected. No page in effect.	SET. The preceding options are mutually exclusive.
Parameter Error	SUBMIT. Within the SUBMIT file of type SUB valid parameters are $0 through $9.
Password Error.	DUMP ERASE GENCOM TYPE. The password is incorrect.
Physical Device Does Not Exist.	DEVICE. The specified physical device is not defined in the system.

PROGRAM INPUT IGNORED.

SUBMIT. This message is preceded by "WARNING". The SUBMIT file contains a line with < and the program does not require additional input.

PUT>

PUT. This prompt occurs when a program requests input while running a PUT FILE [NO ECHO] command.

PUT ERROR: FILE ERASED.

PUT. The PUT output file was erased and could not be closed.

QUIT NOT FOUND

PIP. The string argument to a Q parameter was not found in your input file.

Random Read

SUBMIT. An error occurred when reading the temporary file used by the SUBMIT command.

Read only.

GENCOM SET. The drive or file specified is write-protected.

Read error

TYPE. An error occurred when reading the file specified in the TYPE command. Check the disk and try again.

Reading file HELP.HLP
Reading HELP.HLP index

HELP. An error occurred while reading HELP.HLP. Copy the HELP.HLP file from the system disk.

Table A-1 (continued)

RECORD TOO LONG	
	PIP. A HEX record exceeds 80 characters in a file being copied with the [H] option.
Requires CP/M PLUS.0 or higher.	
	DATE DEVICE DIR ERASE GENCOM HELP INITDIR PIP SET SETDEF SHOW RENAME TYPE. This version of the utility must only be run under CP/M PLUS.0 or higher.
R/O DISK	
	PIP. The destination drive is set to Read-Only and PIP cannot write to it.
R/O FILE	
	PIP. The destination file is set to Read-Only and PIP cannot write to it.
Sort Stack Overflow	
	DIR. There is not enough memory available for the sort stack.
SOURCE FILE READ ERROR:	
	MAC RMAC. The source file cannot be read properly by MAC.
SOURCE FILENAME ERROR:	
	MAC RMAC. The form of the source filename is invalid.

START NOT FOUND

PIP. The string argument to an S parameter cannot be found in the source file.

Symbol Table overflow

XREF. No space is available for an attempted symbol allocation.

Symbol Table reference overflow

XREF. No space is available for an attempted symbol reference allocation.

SYNTAX ERROR:

LIB. The LIB-80 command is not properly formed.

Too many entries in Index Table. Not enough memory

HELP. There is not enough memory available to hold the topic table while creating HELP.HLP.

Topic:
xxxxxx
Not found.

HELP. The topic requested does not exist in the HELP.HLP file. HELP displays the topics available.

Total file size exceeds 64K.

GENCOM. The output file exceeds the maximum allowed.

Try 'PAGE' or 'NO PAGE'

TYPE. The only valid option is PAGE or NO PAGE.

Table A-1 (continued)

Unable to close HELP.DAT.
Unable to close HELP.HLP.

HELP. An error occurred while closing file HELP.HLP or HELP.DAT. There might not be enough disk or directory space on the drive.

Unable to find file HELP.HLP.

HELP. HELP requires HELP.HLP file to operate. Copy it to your default drive from your CP/M PLUS system disk.

Unable to Make HELP.DAT.
Unable to Make HELP.HLP.

HELP. There is not enough space on the disk for HELP.HLP or HELP.DAT or the files are Read-Only.

UNBALANCED MACRO LIBRARY.

MAC RMAC. A MACRO definition was started within a macro library but the end of the file was found in the library before the balancing ENDM was encountered.

UNDEFINED START SYMBOL:

LINK-80. The symbol specified with the G switch is not defined in any of the modules being linked.

UNDEFINED SYMBOLS:

LINK-80. The symbols following this message are referenced but not defined in any of the modules being linked.

UNEXPECTED END OF HEX FILE - filespec

PIP. An end-of-file was encountered before a termination hex record. The hex file without a termination record should be corrected probably by recreating the hex file.

Unrecognized drive.

SHOW. The specified drive is not valid. Valid drives are A to P.

UNRECOGNIZED ITEM:

LINK-80. An unfamiliar bit pattern has been scanned and ignored by LINK-80.

Unrecognized input.

SHOW. The SHOW command line has an invalid format.

Unrecognized option.

GENCOM and SHOW. An option typed in the command line is not valid for the command.

USER ABORTED

PIP. You stopped a PIP operation by pressing CTRL-C.

VERIFY ERROR: - filespec

PIP. When copying with the V option PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer.

Write error

XREF. This message is preceded by filename.XRF and indicates that no disk space is available or no directory space exists on the specified drive.

Writing file: filespec

HELP. An error occurred while attempting to write the file specified by filespec.

Table A-1 (continued)

Wrong Password.	
	SET. The specified password is incorrect or invalid.
?	
	SID. SID has encountered an error.

Appendix B
ASCII and Hexadecimal Conversions

ASCII stands for American Standard Code for Information Interchange. The code contains 96 printing and 32 non-printing characters used to store data on a disk. Table B-1 defines ASCII symbols then Table B-2 lists the ASCII and hexadecimal conversions. The table includes binary decimal hexadecimal and ASCII conversions.

Table B-1. ASCII Symbols

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

Table B-2. ASCII Conversion Table

Binary	*Decimal*	*Hexadecimal*	*ASCII*
0000000	0	0	NUL
0000001	1	1	SOH (CTRL-A)
0000010	2	2	STX (CTRL-B)
0000011	3	3	ETX (CTRL-C)
0000100	4	4	EOT (CTRL-D)
0000101	5	5	ENQ (CTRL-E)
0000110	6	6	ACK (CTRL-F)
0000111	7	7	BEL (CTRL-G)
0001000	8	8	BS (CTRL-H)
0001001	9	9	HT (CTRL-I)
0001010	10	A	LF (CTRL-J)
0001011	11	B	VT (CTRL-K)
0001100	12	C	FF (CTRL-L)
0001101	13	D	CR (CTRL-M)
0001110	14	E	SO (CTRL-N)
0001111	15	F	SI (CTRL-O)
0010000	16	10	DLE (CTRL-P)
0010001	17	11	DC1 (CTRL-Q)
0010010	18	12	DC2 (CTRL-R)
0010011	19	13	DC3 (CTRL-S)
0010100	20	14	DC4 (CTRL-T)
0010101	21	15	NAK (CTRL-U)
0010110	22	16	SYN (CTRL-V)
0010111	23	17	ETB (CTRL-W)
0011000	24	18	CAN (CTRL-X)
0011001	25	19	EM (CTRL-Y)
0011010	26	1A	SUB (CTRL-Z)
0011011	27	1B	ESC (CTRL-[)
0011100	28	1C	FS (CTRL-\)
0011101	29	1D	GS (CTRL-])
0011110	30	1E	RS (CTRL-^)
0011111	31	1F	US (CTRL--)
0100000	32	20	(SPACE)
0100001	33	21	!
0100010	34	22	"
0100011	35	23	#
0100100	36	24	$

Binary	*Decimal*	*Hexadecimal*	*ASCII*
0100101	37	25	%
0100110	38	26	&
0100111	39	27	'
0101000	40	28	(
0101001	41	29	)
0101010	42	2A	*
0101011	43	2B	+
0101100	44	2C	,
0101101	45	2D	-
0101110	46	2E	.
0101111	47	2F	/
0110000	48	30	0
0110001	49	31	1
0110010	50	32	2
0110011	51	33	3
0110100	52	34	4
0110101	53	35	5
0110110	54	36	6
0110111	55	37	7
0111000	56	38	8
0111001	57	39	9
0111010	58	3A	:
0111011	59	3B	;
0111100	60	3C	<
0111101	61	3D	=
0111110	62	3E	>
0111111	63	3F	?
1000000	64	40	@
1000001	65	41	A
1000010	66	42	B
1000011	67	43	C
1000100	68	44	D
1000101	69	45	E
1000110	70	46	F
1000111	71	47	G
1001000	72	48	H
1001001	73	49	I

Table B-2 (continued)

Binary	*Decimal*	*Hexadecimal*	*ASCII*
1001010	74	4A	J
1001011	75	4B	K
1001100	76	4C	L
1001101	77	4D	M
1001110	78	4E	N
1001111	79	4F	O
1010000	80	50	P
1010001	81	51	Q
1010010	82	52	R
1010011	83	53	S
1010100	84	54	T
1010101	85	55	U
1010110	86	56	V
1010111	87	57	W
1011000	88	58	X
1011001	89	59	Y
1011010	90	5A	Z
1011011	91	5B	[
1011100	92	5C	\
1011101	93	5D	]
1011110	94	5E	ˆ
1011111	95	5F	<
1100000	96	60	’
1100001	97	61	a
1100010	98	62	b
1100011	99	63	c
1100100	100	64	d
1100101	101	65	e
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

Binary	*Decimal*	*Hexadecimal*	*ASCII*
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
111000	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

Appendix C
Filetypes

CP/M Plus identifies every file by a unique file specification which consists of a drive specification a filename a filetype and an optional password. The filetype is an optional three-character ending separated from the filename by a period. The filetype generally indicates a special kind of file. The following table lists common filetypes and their meanings.

Table C-1. Common Filetypes

Type	*Meaning*
ASM	Assembly language source file; the CP/M Plus assemblers assemble or translate a type ASM file into machine language.
BAK	Back-up file created by text editor; the editor renames the source file with this filetype to indicate that the original file has been processed. The original file stays on disk as the back-up file so you can refer to it.
BAS	MALLARD BASIC or CBASIC program source file.
COM	8080 executable file.
ERL	Pascal/MT+™ relocatable file.
HEX	Program file in hexadecimal format.
INT	BASIC program intermediate language file.
IRL	Indexed REL file produced by LIB.
LIB	Used by MAC and RMAC for macro libraries. The ED R command reads files of type LIB. The ED X command

	writes files of type LIB. Printable file displayable on console or printer.
OVL	Program overlay file. PL/I-80 compiler overlays files; you can create overlay files with LINK-80.
PAS	Pascal/MT+ source program filetype.
PLI	PL/I-80 source program filetype.
PRL	Page Relocatable file; a file that does not require an absolute segment. It can be relocated in any Page Boundary (256 Bytes).
PRN	Printable file displayable on console or printer.
REL	Relocatable file produced by RMAC and PL/I-80 that can be linked by LINK-80.
SPR	System Page Relocatable file; system files required to generate CP/M Plus such as BNKBDOS.SPR BDOS.SPR BIOS.SPR and RESBDOS.SPR.
SUB	Filetype required for submit file containing one or more CP/M Plus commands. The SUBMIT program executes commands in files of type SUB providing a batch execution mode for CP/M Plus.
SYM	Symbol table file. MAC RMAC and LINK-80 output files of type SYM. SID and ZSID read files of type SYM.
SYS	System file for CP/M Plus.
TEX	Source file for TEX-80™, the Digital Research text formatter.
TOK	Pascal/MT+ intermediate language file.
XRF	Cross-reference file produced by XREF.
$$$	Temporary file.

Appendix D
CP/M Plus Control Character Summary

This appendix describes the use of CTRL characters. Note that CTRL is given by the ALT key on the PCW8256.

CTRL-A	Moves the cursor one character to the left.
CTRL-B	Moves the cursor to the beginning of the command line without having any effect on the contents of the line. If the cursor is at the beginning CTRL-B moves it to the end of the line.
CTRL-C	Terminates the executing program and redisplays the system prompt provided the cursor is at the beginning of the command line. Also if you halt scrolling with CTRL-S you can terminate the program with a CTRL-C.
CTRL-E	Forces a physical carriage return but does not send the command line to CP/M Plus. Moves the cursor to the beginning of the next line without erasing the previous input.
CTRL-F	Moves the cursor one character to the right.
CTRL-G	Deletes the character indicated by the cursor. The cursor does not move.
CTRL-H	Deletes a character and moves the cursor left one character position.
CTRL-I	Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the TAB key.

CTRL-J	Sends the command line to CP/M Plus and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or ENTER or a CTRL-M keystroke.
CTRL-K	Deletes to the end of the line from the cursor.
CTRL-M	Sends the command line to CP/M Plus and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or ENTER or a CTRL-J keystroke.
CTRL-P	Echoes all console activity to the printer. The first time you type CTRL-P CP/M Plus rings a bell at your console. You can use CTRL-P after you halt scrolling with CTRL-S. A second CTRL-P ends printer echo; no bell rings. CTRL-P has no effect if your system does not include a printer.
CTRL-R	Retypes the command line. Places a # at the current cursor location moves the cursor to the next line and retypes any partial command you typed so far.
CTRL-S	Stops screen scrolling. If a display scrolls by too fast for you to read it type CTRL-S. CTRL-Q restarts screen scrolling.
CTRL-U	Discards all the characters in the command line places a # at the current cursor position and moves the cursor to the next line. However you can use a CTRL-W to recall any characters that were to the left of the cursor when you pressed CTRL-U.
CTRL-W	Recalls and displays previously entered command line both at the operating system level and in executing programs if the CTRL-W is the first character entered after the prompt. CTRL-J CTRL-M CTRL-U and RETURN define the command line you can recall. If the command line contains characters CTRL-W moves the cursor to the end of the command line. If you press RETURN CP/M Plus executes the recalled command.
CTRL-X	Discards all the characters left of the cursor and moves the cursor to the beginning of the current line. CTRL-X saves any characters right of the cursor. banked

The PCW8256 has the following default expansion string assignments which are of use within both CP/M Plus commands and Mallard BASIC.

Expansion token	*Expansion string*	*Key combination*
#80	Control-C	STOP
#81	Control-Z	f1
#82	Control-Z	f2 (SHIFT + f1)
#83	Control-Q	f3
#84	Control-Q	f4 (SHIFT + F3)
#85	Control-S	f4
#86	Control-S	f5 (SHIFT + f4)
#87	Control-P	f7
#88	Control-P	f8 (SHIFT + f7)
#89	Control-G	DEL→
#8B	Control-H	CAN
#8C	Control-U	CUT
#8D	Control-W	PASTE
#8E	Control-]	FIND
#8F	Control-F Control-B Control B	EOL
#90	Control-F Control-B	LINE
#91	Control-_	↑
#92	Control-V	+
#93	Control-A	←
#94	Control-F	→ or CHAR
#95	Control-R	RELAY
#96	Control-^	↓
#97	Control-K	ALT + DEL→
#98	Control-\	—
#99	Control-E	ALT + ↓
#9A	Control-X	ALT + ←DEL

Consult your User Guide for further details.

Appendix E
System Control Block

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

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

Table E-1. SCB Fields and Definitions

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

Table E-1 (continued)

Offset	*RW/RO*	*Definition*
12 - 19	RO	Reserved for system use
1A	RW	Console Width. This byte contains the number of columns, characters per line, on your console relative to zero. Most systems default this value to 79. You can set this default value by using the DEVICE utility. The console width value is used by the banked version of CP/M Plus in BDOS function 10, CP/M Plus's console editing input function. Note that typing a character into the last position of the screen, as specified by the Console Width field, must not cause the terminal to advance to the next line.
1B	RO	Console Column Position. This byte contains the current console column position.
1C	RW	Console Page Length. This byte contains the page length, lines per page, of your console. Most systems default this value to 24 lines per page. This default value may be changed by using the DEVICE utility.
ID - 21	RO	Reserved for system use.
22 - 2B	RW	Redirection flags for each of the five logical character devices. If your system's BIOS supports assignment of logical devices to physical devices, you can direct each of the five logical character devices to any combination of up to 12 physical devices. The 16-bit word for each device represents the following: Each bit represents a physical device where bit 15 corresponds to device zero and bit 4 corresponds to device 11. Bits zero through 3 are reserved for system use.

Offset	*RW/RO*	
		You can redirect the input and output logical devices with the DEVICE command.
22 - 23	RW	CONIN Redirection Flag.
24 - 25	RW	CONOUT Redirection Flag.
26 - 27	RW	AUXIN Redirection Flag.
28 - 29	RW	AUXOUT Redirection Flag.
2A - 2B	RW	LSTOUT Redirection Flag.
2C	RW	Page Mode. If this byte is set to zero, some CP/M Plus utilities and CCP built-in commands display one page of data at a time; you display the next page by pressing any key. If this byte is not set to zero, the system displays data on the screen without stopping. To stop and start the display, you can press CTRL-S and CTRL-Q, respectively.
2D	RO	Reserved for system use.
2E	RW	Determines if CTRL-H is interpreted as a rub/del character. If this byte is set to 0, then CTRL-H is a backspace character (moves back and deletes). If this byte is set to 0FFH, then CTRL-H is a rub/del character, echoes the deleted character.
2F	RW	Determines if rub/del is interpreted as CTRL-H character. If this byte is set to 0, then rub/del echoes the deleted character. If this byte is set to 0FF, then rub/del is interpreted as a CTRL-H character (moves back and deletes).

Table E-1 (continued)

Offset	*RW/RO*	*Definition*
30 - 32	RO	Reserved for system use.
33 - 34	RW	Console Mode. This is a 16-bit system parameter that determines the action of certain BDOS Console I/O functions. (See Section 8 and BDOS Function 109, Get/Set Console Mode, for a thorough explanation of Console Mode.)
35 - 36	RO	Reserved for system use.
37	RW	Output delimiter character. The default output delimiter character is $, but you can change this value by using the BDOS Function 110, Get/Set Output Delimiter.
38	RW	List Output Flag. If this byte is set to 0, console output is not echoed to the list device. If this byte is set to 1 console output is echoed to the list device.
39 - 3B	RO	Reserved for system use.
3C - 3D	RO	Current DMA Address. This address can be set by BDOS Function 26 (Set DMA Address). The CCP initializes this value to 0080H. BDOS Function 13, Reset Disk System, also sets the DMA address to 0080H.
3E	RO	Current Disk. This byte contains the currently selected default disk number. This value ranges from 0 - 15 corresponding to drives A - P, respectively. BDOS Function 25, Return Current Disk, can be used to determine the current disk value.
3F - 43	RO	Reserved for system use.

Offset	*RW/RO*	*Definition*
44	RO	Current User Number. This byte contains the current user number. This value ranges from 0 - 15. BDOS Function 32, Set/Get User Code, can change or interrogate the currently active user number.
45 - 49	RO	Reserved for system use.
4A	RW	BDOS Multi-Sector Count. This field is set by BDOS Function 44, Set Multi-sector Count.
4B	RW	BDOS Error Mode. This field is set by BDOS Function 45, Set BDOS Error Mode. If this byte is set to 0FFH, the system returns to the current program without displaying any error messages. If it is set to 0FEH, the system displays error messages before returning to the current program. Otherwise, the system terminates the program and displays error messages. See description of BDOS Function 45, Set BDOS Error Mode, for discussion of the different error modes.
4C - 4F	RW	Drive Search Chain. The first byte contains the drive number of the first drive in the chain, the second byte contains the drive number of the second drive in the chain, and so on, for up to four bytes. If less than four drives are to be searched, the next byte is set to 0FFH to signal the end of the search chain. The drive values range from 0 - 16, where 0 corresponds to the default drive, while 1 - 16 corresponds to drives A - P, respectively. The drive search chain can be displayed or set by using the SETDEF utility.
50	RW	Temporary File Drive. This byte contains the drive number of the temporary file drive. The

Table E-1 (continued)

Offset	*RW/RO*	*Definition*
		drive number ranges from 0 - 16, where 0 corresponds to the default drive, while 1 - 16 corresponds to drives A - P, respectively.
51	RO	Error drive. This byte contains the drive number of the selected drive when the last physical or extended error occurred.
52 - 56	RO	Reserved for system use.
57	RO	BDOS Flags. Bit 7 applies to banked systems only. If bit 7 is set, then the system displays expanded error messages. The second error line displays the function number and FCB information. Bit 6 applies only to nonbanked systems. If bit 6 is set, it indicates that GENCPM has specified single allocation vectors for the system. Otherwise, double allocation vectors have been defined for the system. Function 98, Free Blocks, returns temporarily allocated blocks to free space only if bit 6 is reset.
58 - 59	RW	Date in days in binary since 1 Jan 78.
5A	RW	Hour in BCD (2-digit Binary Coded Decimal).
5B	RW	Minutes in BCD.
5C	RW	Seconds in BCD.
5D - 5E	RO	Common Memory Base Address. This value is zero for nonbanked systems and nonzero for banked systems.
5F - 63	RO	Reserved for system use.

Appendix F
PRL File Generation

PRL Format

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

Table F-1. PRL File Format

Address	*Contents*
0001-0002H	Program size
0004-0005H	Minimum buffer requirements (additional memory)
0006-00FFH	Currently unused, reserved for future allocation
0100H + Program size = Start of bit map	

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

Generating a PRL

The preferred technique for generating a PRL file is to use the CP/M

LINK-80., which can generate a PRL file from a REL relocatable object file. A sample link command is shown.

```
A>link dump[op]
```

Appendix G
SPR Generation

System Page Relocatable, SPR, files are similar in format to PRL files except that SPR files have an origin offset of 0000H (see Appendix F). SPR Files are provided as part of the standard CP/M Plus System: the resident and banked portions of the banked BDOS, named RESBDOS3.SPR and BNKBDOS3.SPR. The BIOS SPR file is named BNKBIOS3.SPR for banked systems.

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

- If LINK-80 is used, the output file of type SPR is specified with the [os] or option. The option is used when linking BNKBIOS3.SPR.
- The code in the SPR is ORGed at 000H rather than 100H.

Appendix H
BDOS Function Summary

Note: . indicates the address of

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

Function Number/Name	*Input Parameters*	*Returned Values*
22 Make File	DE = .FCB	A = Dir Code
23 Rename File	DE = .FCB	A = Dir Code
24 Return Login Vector	none	HL = Login Vector
25 Return Current Disk	none	A = Cur Disk#
26 Set DMA Address	DE = .DMA	A = 00H
27 Get Addr(Alloc)	none	HL = .Alloc
28 Write Protect Disk	none	A = 00H
29 Get R/O Vector	none	HL = R/O Vector
30 Set File Attributes	DE = .FCB	A = Dir Code
31 Get Addr(DPB)	none	HL = .DPB
32 Set/Get User Code	E = 0FFH/ user number	A = Curr User/ 00H
33 Read Random	DE = .FCB	A = Err Code
34 Write Random	DE = .FCB	A = Err Code
35 Compute File Size	DE = .FCB	r0, r1, r2 A = Err Flag
36 Set Random Record	DE = .FCB	r0, r1, r2
37 Reset Drive	DE = Drive Vector	A = 00H
38 Access Drive	none	A = 00H
39 Free Drive	none	A = 00H
40 Write Random with Zero Fill	DE = .FCB	A = Err Code
41 Test and Write Record	DE = .FCB	A = 0FFH
42 Lock Record	DE = .FCB	A = 00H
43 Unlock Record	DE = .FCB	A = 00H
44 Set Multi-sector Count	E = # Sectors	A = Return Code
45 Set BDOS Error Mode	E = BDOS Err Mode	A = 00H
46 Get Disk Free Space number	E = Drive A = Err Flag	Number of Free Sectors
47 Chain to Program	E = Chain Flag	A = 00H
48 Flush Buffers	E = Purge Flag	A = Err Flag
49 Get/Set System Control Block	DE = .SCB PB	A = Returned Byte HL = Returned Word
50 Direct BIOS Calls	DE = .BIOS PB	BIOS Return
59 Load Overlay	DE = .FCB	A = Err Code
60 Call Resident System Extension	DE = .RSX PB	A = Err Code

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

Appendix I
Extended Disk Parameter Blocks

Associated with each (logical) drive is an extended disk parameter block (XDPB). This contains a standard DPB as required by CP/M Plus and information required by the BIOS to support the different formats. It may be patched in order to use different format disks provided that the restrictions detailed below are obeyed.

XDPB structure

```
  bytes 0 . . 16: standard CP/M Plus DPB
  byte  17      : sidedness
                  0 => single sided
                  1 => double sided flip sides
                  2 => double sided up and over
  byte  18      : number of tracks per side
  byte  19      : number of sectors per track
  byte  20      : first sector number
  byte  21, 22  : sector size
  byte  23      : gap length (read/write)
  byte  24      : gap length (format)
  byte  25      : bit 7 multi-track operation
                     1 => multi-track operation
                     0 => single track
                  bit 6 modulation mode
                     1 => MFM mode
                     0 => FM mode
                  bit 5 skip deleted data address mark
                     1 => skip deleted data address mark
                     0 => don't skip deleted address mark
                  bits 4 . . 0 = 0
byte  26      : freeze flag
                #00 => auto-detect disk format
                #FF => don't auto-detect disk format
```

Byte 25 is normally set to #60. Multi-track operation is not recommended.

Setting the freeze flag (byte 26) prevents the BIOS from trying to determine the format of a disk. This should be used when patching an XDPB for a non-standard format.

To find the XDPB for a particular drive use BDOS function 31.

The restriction on patching an XDPB is that the resulting disk structure must lie within the following maximum sizes:

Maximum 2 bit allocation vector	= 91 bytes
Maximum checksum vector	= 32 bytes
Maximum hash table size	= 512 bytes
Maximum sector size	= 512 bytes

This corresponds to a disk of 160 tracks, 9 sectors per track, 512 bytes per sector, 1 reserved track, 128 directory entries, 2K block size. This is the recommended structure for a double sided, double track, double density disk.

The XDPBs for the standard formats are as follows.

PCW8256 Format (type 0)

36	SPT, records per track
3	BSH, block shift
7	BLM, block mask
0	EXM, extent mask
174	DSM, number of blocks – 1
63	DRM, number of directory entries – 1
#C0	AL0, 2 directory blocks
#00	AL1
16	CKS, size of checksum vector
1	OFF, reserved tracks
2	PSH, physical sector shift
3	PHM, physical sector mask
0	single sided
40	tracks per side

9	sectors per track
1	first sector number
512	sector size
42	gap length (read/write)
82	gap length (format)
#60	MFM mode, skip deleted data address mark
0	do auto select format

System Format (PCW8256 and CPC6128)

36	SPT, records per track
3	BSH, block shift
7	BLM, block mask
0	EXM, extent mask
170	DSM, number of blocks – 1
63	DRM, number of directory entries – 1
#C0	AL0, 2 directory blocks
#00	AL1
16	CKS, size of checksum vector
2	OFF, reserved track
2	PSH, physical sector shift
3	PHM, physical sector mask

0	single sided
40	tracks per side
9	sectors per track
#41	first sector number
512	sector size
42	gap length (read/write)
82	gap length (format)
#60	MFM mode, skip deleted data address mark
0	do auto select format

Data Only Format (PCW8256 and CPC6128)

36	SPT, records per track
3	BSH, block shift
7	BLM, block mask
0	EXM, extent mask

179	DSM, number of blocks – 1
63	DRM, number of directory entries – 1
#C0	AL0, 2 directory blocks
#00	AL1
16	CKS, size of checksum vector
0	OFF, reserved tracks
2	PSH, physical sector shift
3	PHM, physical sector mask
0	single sided
40	tracks per side
9	sectors per track
#C1	first sector number
512	sector size
42	gap length (read/write)
82	gap length (format)
#60	MFM mode, skip deleted data address mark
0	do auto select format

Swapping Disks Between CP/M Plus and CP/M 2.2

CP/M 2.2 and AMSDOS system format and data only format disks can be used under CP/M Plus with no further ado.

CP/M Plus disks can be freely used under CP/M 2.2 or AMSDOS provided that:

There is no disk label.
There is no time and date stamping.
There are no passwords.

This is the normal state of a CP/M Plus disk, and will remain as such unless the user explicitly enables any of the above features using INITDIR.COM and/or SET.COM.

If any of the above exist then both CP/M 2.2 and AMSDOS will become confused as to the amount of free space on the disk. Files may still be read but it is recommended that such disks are not written to under CP/M 2.2 or AMSDOS.

INITDIR.COM is used to reformat the directory for time and date stamping. SET.COM is used to create disk labels and enable and create passwords.

Appendix J

BIOS Extended Jumpblock

USERF

In the standard CP/M Plus BIOS jumpblock function 30 "USERF" is reserved for the system implementor. On the CPC6128 and PCW8256 this function is used for calling the firmware and extended BIOS routines in bank 0.

On CPC6128 you can use USERF to access the firmware (see Appendix K).

To find USERF fetch the contents of location 1, this contains the address of function 1 "WBOOT" in the BIOS jumpblock. Add 87 to give the address of the JMP USERF entry. It is a good idea to copy the USERF jumpblock entry into a fixed location and then call this fixed location.

USERF takes the address of the required routine in bank 0 as an inline parameter. The registers AF BC DE HL IX IY are all passed to the routine and returned back to the caller as set by the routine. The alternate register set is preserved throughout the call, it can neither be used to pass parameters nor to return results.

BIOS Jumpblocks

The BIOS has two jumpblocks: the standard CP/M Plus jumpblock and an extended jumpblock. The word at location #0001 contains the address of the WBOOT entry in the standard BIOS jumpblock, function 1. The extended jumpblock starts at #0080 in bank 0 and contains jumps for additional facilities such as physical sector reading and writing.

Since the extended BIOS jumpblock is in bank 0 whereas the user program is in bank 1 it is not possible for an application program to call routines in the extended jumpblock directly, instead the BIOS function USERF must be used.

Disk Driver

```
#0080  JMP  DD INIT            ;initialize disk driver
#0083  JMP  DD SETUP           ;set disk parameters
#0086  JMP  DD READ SECTOR     ;read a sector
#0089  JMP  DD WRITE SECTOR    ;write a sector
#008C  JMP  DD CHECK SECTOR    ;check a sector
#008F  JMP  DD FORMAT          ;format a track
#0092  JMP  DD LOGIN           ;login a disk
#0095  JMP  DD SEL FORMAT      ;select a standard format
#0098  JMP  DD DRIVE STATUS    ;fetch drive status
#009B  JMP  DD READ ID         ;read a sector ID
#009E  JMP  DD L DPB           ;initialize a DPB
#00A1  JMP  DD L XDPB          ;initialize an XDPB
#00A4  JMP  DD L ON MOTOR      ;turn motor on, wait for timeout
#00A7  JMP  DD L T OFF MOTOR   ;set motor off timeout
#00AA  JMP  DD L OFF MOTOR     ;turn motor off
#00AD  JMP  DD L READ          ;read type uPD765A command
#00B0  JMP  DD L WRITE         ;write type uPD765A command
#00B3  JMP  DD L SEEK          ;seek command
```

SIO Driver

```
#00B6  JMP  CD SA INIT         ;initialize SIO channel A
#00B9  JMP  CD SA BAUD         ;set SIO channel A baud rates
#00BC  JMP  CD SA PARAMS       ;fetch SIO channel A parameters
```

Terminal Emulator

```
#00BF  JMP  TE ASK             ;where is the cursor, what screen
                                  size
#00C2  JMP  TE RESET           ;re-initialize the screen
#00C5  JMP  TE STL ASK         ;is the status line enabled?
#00C8  JMP  TE STL ON OFF      ;enable/disable the status line
#00CB  JMP  TE SET INK         ;set the colour(s) for an ink
#00CE  JMP  TE SET BORDER      ;set the colour(s) for the border
#00D1  JMP  TE SET SPEED       ;set the ink flash speed
```

Keyboard

```
#00D4  JMP  KM SET EXPAND       ;set the text for an expansion
                                  token
#00D7  JMP  KM SET KEY          ;set entry(s) for key translation
#00DA  Jmp  KM KT GET           ;get a key token
#00DD  JMP  KM KT PUT           ;put a key token
#00E0  JMP  KM SET SPEED        ;set key repeat speed
```

Misc

```
#00E3  JMP  CD VERSION          ;get version numbers
#00E6  JMP  CD INFO             ;get BIOS system information
```

The PCW8256 additionally has the entry

```
#00E9  JMP  SCR RUN ROUTINE  ;run a routine in screen
                               environment
```

All other entries in the jumpblock are reserved and must not be used.

The action performed by each entry in the extended jumpblock is detailed below, along with the entry and exist conditions of each.

0 DD INIT #0080

Initialize the disk driver.

Action: Initializes the disc driver, resets all disc parameters to their default values. Turns the motor off.

Entry conditions: None.

Exit conditions: AF BC DE HL corrupt. All other registers preserved

Notes: The default disk parameters are:

Motor on timeout	1 sec
Motor off timeout	5 sec
Write current off time	1.75 msec
Head settle time	30 msec
Step rate	12 msec
Head load time	4 msec
Head unload time	480 msec
Non-DMA mode	

Related Entries: DD SETUP

1 DD SETUP #0083

Reset various disk parameters.

Action: Sets the values for the motor on, motor off, write current off and head settle times. Sends a SPECIFY command to the floppy disk controller.

Entry conditions:

HL = address of parameter block in common memory. Format of the parameter block:

bytes 0: motor on timeout in 100 msec units.
bytes 1: motor off timeout in 100 msec units.
byte 2: write current off time in 10 μsec units.
byte 3: head settle time in 1 msec units.
byte 4: step rate time in 1 msec units.
byte 5: head unload delay 32 . . 480 msec in 32 msec units.
byte : bits 7 . . 1: head load delay, bit 0: non-DMA mode (as per uPD765A SPECIFY command).

Exit conditions: AF BC DE HL corrupt. All other registers preserved

Notes: The values given are used for both drives. When using two differing drives use the slower of the two times. A motor off time of zero will never turn the motor off. The default values are:

Motor on timeout	10
Motor off timeout	50
Write current off time	175
Head settle time	30
Step rate	12
Head load time	#0F
Head unload time + non-DMA	#03

2 DD READ SECTOR #0086

Read a sector from disk.

Action: Reads a sector from disk into any bank.

Entry conditions:
- B = CP/M bank
- C = unit
- D = logical track
- E = logical sector
- HL = address of destination buffer in the bank in register B
- IX = address of XDPB in common memory (#C000 . . #FFFF)

Exit conditions: If OK:
- Carry true
- A corrupt

If failed:
- Carry false
- A = reason
  - 0 => drive not ready
  - 2 => seek fail
  - 3 => data error
  - 4 => no data
  - 5 => missing address mark
  - 8 => media changed

Always:
- Other flags BC DE HL corrupt. All other registers preserved

Notes: In the event of errors this routine trys and retrys a total of 15 times as follows:

T T T R T T T T I T T T T R T T T T

where T means try, R realign, I move to innermost track.

The "media changed" error means that the sector numbers on the disk are different from those specified in the XDPB.

Related entries: DD WRITE SECTOR and DD CHECK SECTOR.

3 DD WHITE SECTOR #0089

Write a sector to disk.

Action: Writes a sector to disk from any bank.

Entry conditions:

B = CP/M bank
C = unit
D = logical track
E = logical sector
HL = address of source buffer in bank in register B
IX = address of XDPB in common memory (#C000 . . #FFFF)

Exit conditions: If OK:
Carry true
A corrupt

If failed:
Carry false
A = reason
0 => drive not ready
1 => write protected
2 => seek fail
3 => data error
4 => no data
5 => missing address mark
8 => media changed

Always:
    Other flags BC DE HL corrupt. All other registers preserved.

Notes: In the event of errors this routine trys and retrys a total of 15 times as follows:

T T T R T T T T I T T T T R T T T T

where T means try, R realign, I move to innermost track.

The "media changed" error means that the sector numbers on the disk are different from those specified in the XDPB.

Related entries: DD READ SECTOR and DD CHECK SECTOR.

4 DD CHECK SECTOR #008C

Check that a sector on disk is the same as one in memory.

Action: Compares a sector on the disk with a store copy.

Entry conditions:

B	= CP/M bank
C	= unit
D	= logical track
E	= logical sector
HL	= address of source buffer in bank in register B
IX	= address of XDPB in common memory (#C000 . . #FFFF)

Exit conditions: If OK and the sector on disk = sector in store:
    Carry true zero false
    A corrupt

If OK and the sector on disk <> sector in store:
    Carry true zero false
    A corrupt

If failed:
- Carry false
- A = reason
  - 0 => drive not ready
  - 2 => seek fail
  - 3 => data error
  - 4 => no data
  - 5 => missing address mark
  - 8 => media changed

Always:
Other flags BC DE HL corrupt. All other registers preserved.

Notes: In the event of errors this routine trys and retrys a total of 15 times as follows:

T T T R T T T T I T T T T R T T T T

where T means try, R realign, I move to innermost track.

The "media changed" error means that the sector numbers on the disk are different from those specified in the XDPB.

Related entries: DD READ SECTION and DD WRITE SECTION.

---

5 DD FORMAT #008F

Format a track.

Action: Formats a track using a header information buffer in any bank.

Entry conditions:

B	= CP/M bank
C	= unit
D	= logical track
E	= filler byte, usually #E5
HL	= address of header information buffer in bank in register B
IX	= address of XDPB in common memory (#C000. .#FFFF)

Format of header information:

> sector entry for first sector
> sector entry for second sector
> . . .
> sector entry for last sector

Sector entry format:

> byte 0: track number
> byte 1: head number
> byte 2: sector number
> byte 3: log 2 (sector size) – 7

Exit conditions: If OK:
> Carry true
> A corrupt

If failed:
> Carry false
> A = reason
> > 0 => drive not ready
> > 1 => write protected
> > 2 => seek fail
> > 3 => data error
> > 4 => no data
> > 5 => missing address mark
> > 8 => media changed

Always:
> Other flags BC DE HL corrupt. All other registers preserved

Notes: In the event of errors this routine trys and retrys a total of 15 times as follows:

```
  T T T R T T T T I T T T T R T T T T
```

where T means try, R realign, I move to innermost track.

The "media changed" error means that the sector numbers on the disk are different from those specified in the XDPB.

6 DD LOGIN #0092

Login a disk.

Action: Attempts to determine the format of a disk. If successful, initializes an XDPB. Does not affect or consider the freeze flag.

Entry conditions:

```
    C   = unit
    IX  = address of XDPB in common memory (#C000 . . #FFFF)
```

Exit conditions: If OK:

```
    Carry true
    A  = disk type
          0 => PCW8256 format (type 0)
          1 => system format
          2 => data only format
          other values as read from disk specification
  DE = size of 2 bit allocation vector
  HL = size of hash table
  XDPB initialized
```

If failed:

```
    Carry false
    A = reason
        0 => drive not ready
        2 => seek fail
        3 => data error
        4 => no data
        5 => missing address mark
        6 => bad format
    DE HL corrupt
    XDPB corrupt
```

Always:

```
    Other flags BC corrupt. All other registers preserved
```

Notes: In the event of errors this routine trys and retrys a total of 15 times as follows:

T T T R T T T T I T T T T R T T T T

where T means try, R realign, I move to innermost track.

Related entries: DD SEL FORMAT.

7 DD SEL FORMAT #0095

Select a standard disk format.

Action: Initializes an XDPB for the required format regardless of the actual disk format. Normally the BIOS automatically determines the format of a disk when it is logged in, but for programs such as disk formatters it is necessary to pre-set the format. Does not affect or consider the freeze flag.

Entry conditions:

```
A   = format required
      0 => PCW8256 format, type 0
      1 => system format
      2 => data only format
IX  = address of XDPB in common memory (#C000 . . #FFFF)
```

Exit conditions: If OK:

```
Carry true
A   = disk type
        0 => PCW8256 format, type 0
        1 => system format
        2 => data only format
DE = size of 2 bit allocation vector
HL = size of hash table

XDPB initialized
```

If failed:
    Carry false
    A = reason
        6 => bad format
  DE HL corrupt

    XDPB corrupt

Always:
    Other flags BC corrupt. All other registers preserved

Notes: This routine will only fail if an illegal disk type is given.

Related entries: DD LOGIN.

---

8 DD DRIVE STATUS #0098

Fetch the drive status.

Action: Fetch the drive status: ready, write protected, etc.

Entry conditions: C = bits 0,1: unit, bit 2: head

Exit conditions:
    A = status
        bit 7   : undefined
        bit 6   : write protected
        bit 5   : ready
        bit   4: track 0
        bit 3   : undefined
        bit 2   : head
        bit 1,0: unit

F HL corrupt. All other registers preserved

Notes: For unit 1 if bit 6 = 0, and bit 5 = 0 then the drive is not fitted.

9 DD READ ID #009B

Read a sector ID.

Action: Reads the ID from the first sector found.

Entry conditions:

C = unit
D = logical track
IX = address of XDPB in common memory (#C000 . . #FFFF)

Exit conditions: If OK:
Carry true
A = sector number from ID

If failed:
Carry false
A = reason
0 => drive not ready
2 => seek fail
3 => data error
4 => no data
5 => missing address mark

Always:
HL = address of results buffer in common memory

Format of results buffers

byte 0: number of bytes received
bytes 1 . . : results

F BC DE corrupt. All other registers preserved

Notes: In the event of errors this routine trys and retrys a total of 15 times as follows:

T T T R T T T T I T T T T R T T T T

where T means try, R realign, I move to innermost track.

10 DD L DPB #009E

Initialize a standard DPB.

Action: This routine initializes a standard CP/M Plus Disk Parameter Block for a given disk format. The routine is for PCW8256/8512 only: on CPC6128 it returns the fail condition (see below).

Entry conditions:

HL = address of source disk specification in common memory (#C000 . . #FFFF)
IX = address of destination DPB in common memory (#C000 . . #FFFF)

Exit conditions: If OK:
Carry true
A = disk type

If failed:
Carry false
A = reason
6 => bad format

Always:
Other flags BC DE HL corrupt. All other registers preserved

Notes: The disk format is specified as follows:

Byte 0: disk type
Byte 1: sidedness
0 => single sided
1 => double sided, flip sides
2 => double sided, up and over
Byte 2: number of tracks per side
Byte 3: number of sectors per track
Byte 4: Log2 (sector size) – 7
Byte 5: number of reserved tracks
Byte 6: Log2 (block size) – 7
Byte 7: number of directory blocks
Byte 8: gap length (read/write)
Byte 9: gap length (format)

Related entries: DD L XDPB.

11 DD L XDPB #00A1

Initialize an XDPB.

Action: This routine initializes an eXtended Disk Parameter Block for a given disk format. The routine is for PCW8256/8512 only: on CPC6128 it returns the fail conditon (see below).

Entry conditions:

HL = address of source disk specification in common memory (#C000 . . #FFFF)
IX = address of destination XDPB in common memory (#C000 . . #FFFF)

Exit conditions: If OK:
Carry true
A = disk type

If failed:
Carry false
A = reason
6 => bad format

Always:
Other flags BC DE HL corrupt
All other registers preserved

Notes: The disk format is specified as follows:

Byte 0: disk type
Byte 1: sidedness
0 => single sided
1 => double sided, flip sides
2 => double sided, up and over
Byte 2: number of tracks per side
Byte 3: number of sectors per track
Byte 4: Log2 (sector size) – 7
Byte 5: number of reserved tracks
Byte 6: Log2 (block size) – 7
Byte 7: number of directory blocks
Byte 8: gap length (read/write)
Byte 9: gap length (format)

Related Entries: DD L DPB.

12 DD L ON MOTOR #00A4

Turn the motor on.

Action: If the motor is off then turn it on, wait for the motor on timeout.

Entry conditions: None.

Exit conditions: All registers and flags preserved.

Related entries: DD L T OFF MOTOR and DD L OFF MOTOR.

13 DD L T OFF MOTOR #00A7

Start the motor off timeout.

Action: Starts the motor off timeout, after which the motor will be turned off. Does not wait for the timeout.

Entry conditions: None.

Exit conditions: All registers and flags preserved.

Related entries: DD L ON MOTOR and DD L OFF MOTOR.

14 DD L OFF MOTOR #00AA

Turns the motor off.

Action: Turns the motor off, kills the motor ticker if any.

Entry conditions: None.

Exit conditions: AF BC DE HL corrupt. All other registers preserved.

Related entries: DD L ON MOTOR and DD L T OFF MOTOR.

15 DD L READ #00AD

uPD765A read type command driver.

Action: Low level interface for the uPD765A floppy disk driver. Use for "READ DATA", "READ DELETED DATA", "READ A TRACK". Sends required commands to the uPD765A, deals with bank switching, fetches results. Motor must be running.

Entry conditions:

HL = address of parameter block in common memory (#C000 . . #FFFF)

Format of parameter block:

byte 0	: CP/M bank
byte 1,2	: address of buffer
byte 3,4	: number of bytes to transfer
byte 5	: number of uPD765A command bytes
byte 6 . .	: command bytes

Exit conditions:

HL = address of results buffer in common memory

Format of results buffers

byte 0	: number of results bytes received
bytes 1 . .	: results

AF BC DE corrupt. All other registers preserved.

Notes: Detailed knowledge of the uPD765A is required in order to use this routine.

Related entries: DD L WRITE.

16. DD L WRITE #00B0

uPD765A write type command driver.

Action: Low-level interface for the uPD765A floppy disk driver. Use for "WRITE DATA", "WRITE DELETED DATA", "FORMAT A TRACK", "SCAN EQUAL", "SCAN LOW OR EQUAL", "SCAN HIGH OR EQUAL". Sends required commands to the uPD765A, deals with bank switching, fetches results. Motor must be running.

Entry conditions:

HL = address of parameter block in common memory

Format of parameter block:

byte 0	: CP/M bank
byte 1,2	: address of buffer
byte 3,4	: number of bytes to transfer
byte 5	: number of uPD765A command bytes
byte 6 . .	: command bytes

Exit conditions:

HL = address of results buffer in common memory

Format of results buffers

byte 0	: number of results bytes received
bytes 1 . .	: results

AF BC DE corrupt. All other registers preserved

Notes: Detailed knowledge of the uPD765A is required in order to use this routine.

Related entries: DD L READ.

17 DD L SEEK #00B3

Seek to required track.

Action: Realigns if required, seeks to given track. Motor must be running.

Entry conditions:

C	= bits 0,1: unit, bit 2: head
D	= physical track
IX	= address of XDPB in common memory

Exit conditions: If OK:
Carry true
A corrupt

If failed:
Carry false
A = reason
0 => not ready
2 => seek fail

Always:
Zero true. Other flags B IY corrupt. All registers preserved.

Notes: Will try 10 times to seek or realign before returning an error. This routine is not normally required since "DD READ SECTOR", "DD WRITE SECTOR", "DD CHECK SECTOR" and "DD FORMAT" all perform their own seeks.

18 CD SA INIT #00B6

Initialize SIO for CP/M Plus with version Number less than 1.4.

Action: Initialize SIO channel A.
Entry conditions:
A = mode
#00 => no handshake
#FF => handshake

D = number of stop bits
   0 => 1
   1 => 1.5
   2 => 2
E = parity
   0 => none
   1 => odd
   2 => even
H = number of receive data bits 5, 6, 7 or 8
L = number of transmitter data bits 5, 6, 7 or 8

Exit conditions: AF BC DE HL corrupt. All other registers preserved.

Notes: The parameters are not validated; silly values will give silly results.

This is supported by a new (upwards compatible) interface to the extended BIOS jumpblock routine __SA__INITY:

---

18 CD__SA__INIT **#00B6**

Initialise SIO & set control signals. Only for use with CP/M version 1.4 or greater.

Action: Initialise mode of SIO channel. Raise or lower RTS or DTR on channel A.

Entry conditions:

A = selector
   Mode selection & complete SIO reset:
   #00 (= –0 –0): no handshake, no interrupt mode
   #FF (= –1 –0): handshake, interrupt mode
   #FE (= –0 –2): no handshake, no interrupt mode
   #FD (= –1 –2): handshake, interrupt mode
   Command (does not alter mode or reset SIO)

#80	: drop DTR
#7F	: raise DTR
#7E	: drop RTS
#7D	: raise RTS

D = number of stop bits (not required for commands).
 0 => 1
 1 => 1.5
 2 => 2

E = parity (not required for commands).
 0 => none
 1 => odd
 2 => even

H = number of receive data bits (not required for commands)
 5 . . bits

L = number of transmit data bits (not required for commands)
 5 . . 8 bits

Exit conditions: AF BC DE HL corrupt. All other registers preserved.

Notes: The parameters are not validated, silly values will give silly results. The commands may be issued at any time, but remember that the SIO driver (and interrupt service) routine may themselves raise or drop DTR.

The SIO can now be run with receive interrupts enabled and explicit raising or lowering of the control signals is now supported. The SIO can now be run in four modes:

a) No interrupt, No handshake
DTR & RTS true

Input:	polls "data available"
Output:	polls "Tx buffer empty"

b) No interrupt, handshake
RTS true

Input status:	raises DTR if no character available
Input:	raises DTR if no character available polls "data" available drops DTR when character is read
Output:	polls "Tx all sent" and CTS

c) Interrupt, no handshake
   DTR and RTS true
   interrupts on character reception (including errors)
     reads character into buffer if there is room
     otherwise disarms interrupts

Input status:	checks for characters in interrupt buffer
Input:	polls for character in interrupt buffer raises DTR when character is read & re-arms interrupt
Output:	polls "Tx buffer empty"

d) Interrupt, handshake
   RTS true
   raises DTR initially
   interrupts on character reception (including errors)
   reads character SIO if room in buffer
     otherwise disarms interrupts and drops DTR

Input status:	checks for characters in interrupt buffer
Input:	polls for character in interrupt buffer raises DTR when character is read & re-arms interrupt
Output:	polls "Tx all sent" and CTS

19 CD SA BAUD #00B9

Set the baud rates for channel A of the SIO.

Action: Sets the receiver and transmitter baud rates for SIO channel A.

Entry conditions:

H	= encoded receiver baud rate
L	= encoded transmitter baud rate

Exit conditions: AF BC DE HL corrupt. All other registers preserved.

Notes: The parameters are not validated; silly values will give silly results. Does not affect the CP/M Plus character I/O table.

20 CD SA PARAMS #00BC

Get the current parameters for channel A of the SIO.

Action: Fetches the current mode, parity, baud rates, etc, for SIO channel A.

Entry conditions: None.

Exit conditions:

```
A   = mode
      #00  => no handshake
      #FF => handshake
B   = receiver encoded baud rate
C   = transmitter encoded baud rate
D   = stop bits
      0 => 1
      1 => 1.5
      2 => 2
E   = parity
      0 => none
      1 => odd
      2 => even
H   = receiver data bits 5, 6, 7 or 8
L   = transmitter data bits 5, 6, 7 or 8
```

All other registers preserved.

21 TE ASK #00BF

Get the current viewport position and size and cursor position.

Action: Fetches the current viewport position and size and cursor position.

Entry conditions: None.

Exit conditions:

For PCW8256/8512

B	= top row of the viewport in physical screen coordinates
C	= left column of the viewport in physical screen coordinates
D	= height of viewport – 1
E	= width of viewport – 1
H	= cursor row in viewport coordinates
L	= cursor column in viewport coordinates

All other registers preserved.

For CPC6128

D	= bottom row of screen
E	= right column of screen
H	= cursor row
L	= cursor column

All other registers preserved.

Notes: The top row is row 0, the left column is column 0. Screen size is terminal emulators screen, not physical size. Size depends on whether 24 x 80 mode and status line are enabled.

22 TE RESET #00C2

Re-initialize the terminal emulator.

Action: Re-initializes the terminal emulator: clears the screen, homes and enables the cursor. For use by programs which have written to the screen by means other than using the CRT device.

Entry conditions: None.

Exit conditions: AF BC DE HL corrupt. All other registers preserved.

23 TE STL ASK #00C5

Is the status line enabled?

Action: Asks if the status line is enabled.

Entry conditions: None.

Exit conditions: If enabled:
    Zero false

If disabled:
    Zero true

Always:
    Carry false
    A corrupt
    All other registers preserved

Related entries: TE STL ON OFF.

24 TE STL ON OFF #00C8

Enable/disable the status line.

Action: Enables or disables the status line. Disabling the status line gives an extra line to the terminal emulator. When disabled status line messages are sent to the CONOUT: device.

Entry conditions:

```
A   = enable/disable
      #00 => disable
      #FF => enable
```

Exit conditions: F BC DE HL corrupt. All other registers preserved.

Related Entries: TE STL ASK.

25 TE SET INK #00CB

Set the colours in which to display an ink.

Action: Set which two colours will be used to display an ink. If the two colours are the same then the ink will remain a steady colour. If the two colours are different then the ink will alternate between these colours.

Entry conditions:

A contains an ink number
B contains the first colour
C contains the second colour

Exit conditions: AF BC DE HL corrupt. All other registers preserved.

Notes: The ink number is masked with #0F to make sure it is legal. The colours are masked with #3F and the resulting value is treated as three 2 bit numbers each specifying the intensity of one of the three primary colours. Bits 0,1 for blue, bits 2,3 for red and bits 4,5 for green. On the CPC 6128 the three levels of intensity is mapped as follows:

Colour parameter: 0123 CPC 6128 intensity: 0112

If ink 0 is specified then the border is also changed to the same colours.

On the PCW8256 there are only two inks, 0 and 1, and two colours, black and "white". If colour of ink 0 is greater than the colour of ink 1 then the whole screen is displayed with black characters on "white" background, otherwise "white" characters on a black background.

26 TE SET BORDER #00CE

Set the colours in which to display the border.

Action: Set which two colours will be used to display the border. If the two colours are the same then the border will remain a steady colour. If the two colours are different then the border will alternate between these colours.

Entry conditions: B contains the first colour. C contains the second colour

Exit conditions: AF BC DE HL corrupt. All other registers preserved.

Notes: For the PCW8256 the colours for this routine are bit-significant values:

bit 7 => inverse video, black characters "white" background
bit 6 => border covers whole screen
bits 5 . . 0 ignored

The colours are masked with #3F and the resulting value is treated as three 2 bit numbers each specifying the intensity of one of the three primary colours. Bits 0,1 for blue, bits 2,3 for red and bits 4,5 for green.

On the CPC6128 the three levels of intensity are mapped as follows:

Colour parameter:	0 1 2 3
CPC6128:	0 1 1 2

The border can also be changed by TE SET INK.

27 TE SET SPEED #00D1

Set the flash period.

Action: Set for how long each of the two colours for the inks and the border are to be displayed on the screen. These settings apply to all inks and the border.

Entry conditions:

H contains the period for the first colour
L contains the period for the second colour

Exit conditions: AF HL corrupt. All other registers preserved.

Notes: The flash periods are given in frame flybacks (1/50 or 1/60 of a second). A period of 0 is taken to mean a period of 256.

The default setting for the flash periods is 10 frame flybacks (1/5 or 1/6 of a second).

The new flash periods are not used immediately but when the inks next flash.

On the PCW8256 the whole screen flashes, not individual characters.

28 KM SET EXPAND #00D4

Set an expansion string.

Action: Set the expansion string associated with an expansion token.

Entry conditions:

B contains the expansion token for the expansion to set
C contains the length of the string
HL contains the address of the string in common memory
(#C000 . . #FFFF)

Exit conditions: If the expansion is OK:
Carry true

If the string was too long or the token was invalid:
Carry false

Always:
Other flags A BC DE HL corrupt
All other registers preserved

Notes: The characters in the string are not expanded (or otherwise dealt with). It is therefore possible to put any character into an expansion string.

If there is insufficient room in the expansion buffer for the new string then no change is made to the expansions.

If the string is currently being used to generate characters then the unread portion of the string is discarded. The next character will be read from the key buffer.

29 KM SET KEY #00D7

Set entry(s) in key translation table(s).

Action: Set what character or token a key will be translated to when shift, alt, CTRL on CPC6128 shift and alt, extra, or none of these is pressed.

Entry conditions:

B contains the new translation
C contains a key number
D contains bit mask indicating which table, or tables, are to be changed
bit 0 => normal translation, neither shift, alt nor extra pressed
bit 1 => shift translation
bit 2 => alt translation
bit 3 => shift and alt translation
bit 4 => extra translation
other bits ignored

Exit conditions: AF DE, BC and HL corrupt. All other registers preserved.

Notes: If the key number is invalid then no action is taken.

Most values in the table are treated as characters and are passed back to the user. However, there are certain special values:

#80 . . #9E	are the expansion tokens and are expanded to character strings
#9F	is the ignore token and means the key should be thrown away.

For the CPC6128 these are:

#80 . . #9F	are the expansion tokens and are expanded to character strings.
#FD	is the caps lock token and causes the caps lock to turn on if it is off and vice versa.
#FE	is the shift lock token and changes the shift state on/off.
#FF	is the ignore token.

30 KM KT GET #00DA

Get a key token.

Action: Try to fetch a key token from the keyboard.

Entry conditions: None.

For PCW8256/8512.

Exit conditions: If got a token:
    Carry true
    C = key number

If not got a token:
    Carry false
    C corrupt

Always:
    B = shift state
        bit 0 => not defined
        bit 1 => extra
        bit 2 => caps lock
        bit 3 => repeat
        bit 4 => num lock
        bit 5 => shift
        bit 6 => shift lock
        bit 7 => alt

    Other flags A corrupt. All other registers preserved.

For CPC6128

Exit conditions: If got a token:

    Carry true
    C = character
    B = 0

If not got token:
    Carry false
    BC corrupt

Always:
    Other flags and A corrupt. All other registration preserved.

Notes: The shift state (for PCW8256/8512) only defines which locks and/or shift keys were pressed. The repeat bit indicates that the key was generated by a repeat.

31 KM KT PUT #00DD

Put a key token. No effect on CPC6128.

Action: Insert a key token into the keyboard buffer so that the next "KM KT GET" will fetch it.

Entry conditions:

B = shift state
    bit 0 => not defined
    bit 1 => extra
    bit 2 => caps lock
    bit 3 => repeat
    bit 4 => num lock
    bit 5 => shift
    bit 6 => shift lock
    bit 7 => alt
C = key number

Exit conditions: All registers and flags preserved

Notes: More than one key token may be put, however, buffer overflow is not reported and will result in key tokens being lost. The buffer is at least 10 tokens long. If a great deal of text is to be generated by KM KT PUT the use of expansion tokens is recommended.

To change either the caps lock or num lock states, call "KM KT PUT" with the required state with an innocuous key number, then call "KM KT GET" to remove it. The shift lock state cannot be changed in this way as it is hardware-controlled.

32 KM SET SPEED #00E0

Set key start up delay and repeat speed.

Action: Set the time before keys first repeat (start up delay) and the time between repeats (repeat speed).

Entry conditions:

H contains the new start up delay
L contains the new repeat speed

Exit conditions:

AF corrupt. All other registers preserved.

Notes: Both delays are given in scans of the keyboard. The keyboard is scanned every fiftieth of a second.

A start up delay or repeat speed of 0 is taken to mean 256.

The default start up delay is 30 scans (0.6 sec) and the default repeat speed is 2 scans (0.04 sec or 25 characters a second).

Note that a key is prevented from repeating (by the key scanner) if the key buffer is not empty. Thus the actual repeat speed is the slower of the supplied repeat speed and the rate at which characters are removed from the buffer. This is intended to prevent the user from getting too far ahead of a program that is running sluggishly.

The start up delay and repeat speed apply to all keys on the keyboard that are set to repeat.

33 CD VERSION #00E3

Get version numbers.

Action: Fetches matchine type, BIOS version numbers and machine specific version numbers.

Entry conditions: None.

Exit conditions:

```
A = machine
      0 => CPC6128
      1 => PCW8256
B = BIOS major version number
C = BIOS minor version number
DE reserved
HL = machine specific version number
        CPC6128 =>
        PCW8256 => not defined

        H = ROM version number
        L = ROM mark number
```

All other registers preserved

---

34 CD INFO #00E6

Get BIOS system information.

Action: Fetches BIOS system information, number of disk drives, address of buffer table, number of memory blocks, etc.

Entry conditions: None.

Exit conditions

```
A   = number of disk drives
        #00  => 1 disk drive
        #FF => 2 disk drives
B   = number of memory blocks (8 on CPC6128)
C   = serial interface status
        #00  => not fitted
        #FF => fitted
HL = address of buffer table in common memory
          (#C000 . . #FFFF)
```

Formant of buffer table

    entry for buffer area 0
    entry for buffer area 1
    . . .
    #FF

Entry format:

    byte 0 : CP/M bank
    byte 1,2: start address
    byte 3,4: size in bytes

DE corrupt. All other registers preserved.

Notes: The buffer table gives details of where the data and directory buffer areas are.

35 SCR RUN ROUTINE #00E9

Runs a routine in the screen environment. (8256/8512 only)

Action: Switches memory blocks 7, 2, 1, 0 into context, these blocks contain the character matrix RAM, the roller RAM and the screen RAM. Then calls the supplied routine. On exit the memory is restored to its original state.

Entry conditions:

BC = address of routine to call, in common memory (#C000 . . #FFFF)
AF ~~DE~~ HL IX IY as required by routine

Exit conditions: AF DE HL IX IY as returned by called routine. BC corrupt. All other registers preserved.

Notes: The screen is 720 pixels wide and 256 pixels high. Let (0,0) be the top left corner; (0,719) the top right corner; (255,0) bottom left corner; and (255,719) the bottom right corner.

The character matrix RAM is at #B800 and has the following format:

byte 0:	character 0
byte 8:	character 1
. . .	
byte 2040:	character 255

Each character entry has the following format:

byte 0:	pixel row 0
byte 1:	pixel row 1
. . .	
byte 7:	pixel row 7

Each pixel row has the following format:

bit 0:	pixel column 7
bit 1:	pixel column 6
. . .	
bit 7:	pixel column 0

The roller RAM is at #B600 and has the following format:

bytes 0, 1:	packed address of pixel row 0
bytes 2, 3:	packed address of pixel row 1
. . .	
bytes 510, 511:	packed address of pixel row 255

Each pixel row has the following format:

byte 0:	pixel columns 0 . . 7
byte 8:	pixel columns 8 . . 15
. .	
byte 712:	pixel columns 712 . . 719

Each pixel column byte has the following format:

bit 0:	pixel column 7
bit 1:	pixel column 6
. . .	
bit 7:	pixel column 0

Roller RAM on PCW8256: The PCW8256 screen addressing is set up by CP/M Plus so that each row of characters occupies 720 bytes of memory organised as follows:

	Col 0	Col 1	. . .	Col 89
Top row of char.	Byte 0	Byte 8		Byte 712
	Byte 1	Byte 9		Byte 713
	Byte 6	Byte 14		Byte 718
Bottom row of char.	Byte 7	Byte 15		Byte 719

The entire screen requires around 23K of RAM, which is drawn from two 16K banks of RAM.

The Roller RAM contains one 16-bit entry for each line of pixels. This entry is actually a 17 bit address compressed into 16 bits by missing out bit 3 which must always be zero. The 17 bit address comprises 3 bits which specify which 16K block – from the first 128K of memory – and 14 bits which is the address within the 16K block of the byte specifying the first 8 pixels of the line of pixels. The next 8 pixels come from the byte 8 bytes further through the block. It is therefore possible, as shown above, to arrange pixel lines in groups of 8 so that a character is stored in 8 consecutive bytes, allowing text to be placed on the screen with single block-move instructions. This arrangement of memory effectively makes seven of every eight Roller RAM entries irrelevant for our purposes of writing to the screen memory, as we can always predict their contents. The entries have to exist and be legal, however, because they are read by the VDU hardware. (It is possible to roll the screen, or produce other effects, by re-writing the roller RAM, but this is not recommended as any messages output by CP/M. Including for example the error and status displays on line 31 – will be disturbed). The simplest algorithm for unpacking the 17-bit address makes the valid assumption that the block numbers are very simply related to Z80 addresses, because the CP/M Plus environment arranges for the 16K block of memory called Block 1 to reside in the address range 4000H-7FFFH and for the 16K block of memory called Block 2 to reside in the address range 8000H-BFFFH when the bank switching is set to the "Screen Environment".

Hence it is possible to take the first in a group of 8 of these word addresses and simply shift it one position to the left (i.e. double it). This will give the absolute address in memory of the start of that pixel line.

If it cannot be assumed that the screen is in its default state (i.e. first entry of a group of eight is on an eight byte boundary and the eight entries are consecutive addresses), then, to find the true address: Take the contents of the roller RAM entry, Mask out the bottom 3 bits and move the resultant word one bit to the left. Then replace the bottom three bits. The following is a routine that will achieve this. On entry HL points to a buffer in common memory that contains the data for a number of characters (or groups of 8 character wide pixel masks if it is easier to look at it that way). The H register holds the (character) line number on the screen to which the information should be moved.

The following is a routine which should be moved into common memory (#C000 – #FFFF) and later executed by SCR RUN ROUTINE.

Entry conditions:

```
     hI = address of data for one line of characters (in common memory)
     e  = line number

     roller   equ    0B00h      ;address of roller RAM
     linelen  equ    720        ;number of bytes for a line of characters

     docommon:
              push   hL
              mvi    h,0 LD H,0
              mov    l,e LD L,E ;hl = e
              dad    h ADD HL,HL ;hl = 2#e
              dad    h [illegible] ;hl = 4#e
dad [illegible] h    ;hl = 8#e
              dad    h [illegible] ;hl = 16#e
              lxi    d,roller LD HL,(R[illegible])
              dad    d ADD HL,[illegible] ;hl = roller + 16#e
              mov    e,m LD E,(HL)
              inx    h INC HL
              mov    d,m LD D,(HL) ;de = encoded address

              mov    a,e LD A,E
              ani    7 AND 7    ;also clears carry
              mov    l,a LD L,A ;l = 3 lsb
              mov    a,e LD A,E
              ral    RLA        ;lp8 to carry
```

```
mov     e,a [illegible];lp7-4/lp2-lp0/carry
mov     a,d [illegible]
ral  [illegible]        ;lsb = lp8
mov     d,a [illegible]
mov     a,e [illegible]
ani     0f0H        ;lp7-lp4 [illegible]
ora     1  [illegible]     ;remask with original bottom 3 bits
mov     e, a [illegible];de now holds screen address

pop     h   Pop [illegible]
lxi     b,linelen [illegible]
ldir
ret
```

Appendix K
CPC6128 Firmware Calls

Rules and Restrictions

Using USERF stops an application program from being portable.

Only a subset of firmware routines may be called, they are listed below.

A firmware routine can only be called by using USERF.

Some of the firmware routines require the address of a parameter block. This parameter block must be in common memory, i.e. #C000 and above.

If a firmware routine returns an address this will be an address in bank 0. It is, therefore, likely to be only useful to an application program as a parameter to a further firmware routine.

The upper ROM must not be enabled.

The screen is at #4000 in bank 0, it must not be moved.

The terminal emulator (CRT device) uses the TXT and SCR routines. If an application program also wishes to use these or the GRA routines certain steps must be taken:

> Before using any of the TXT, GRA or SCR routines the terminal emulator's cursor must be disabled by sending ESC f. This is because the cursor is turned on by a ticker which could go off whilst the application program was already in the firmware giving upredictable results.
>
> If the status line is enabled all status line messages will be displayed on the status line regardless of the current CONOUT: device. If the status line is disabled status line messages are sent to the CONOUT: device. Thus to avoid status line messges appearing on the screen disable the

status line and redirect the CONOUT: device away from the CRT device. Alternatively avoid any action which could cause a status line message such as disk errors or selecting logical drive B: on a single drive system.

Device redirection can be performed using the DEVICE.COM utility or by writing to the indirection vectors in the System Control Block (SCB) using BDOS function 49.

If it is required to use the CRT device as well as the TXT, SCR or GRA routines then any mode changing should be done by sending Esc 3 mode to the CRT device, this informs the terminal emulator of the size of the screen etc. Before calling any BIOS or BDOS routine which could cause characters to be sent to the CRT device ensure that:

The cursor position is where the CRT device last had it.
The window covers the whole screen.
Inverse video is as the CRT device last had it.

The status line may still be used if required.

When finished restore the screen to its original state using TE RESET (or by other means) and, if necessary, redirect the CONOUT: device to the CRT.

Summary of Firmware Calls and Restrictions

In this summary "OK" means that the entry can be called using USERF without any more ado. However, an application should take care to restore the machine back to the state in which it found it, otherwise the BIOS screen and keyboard drivers may become confused.

Where addresses are required to be in common memory this is indicated by "Entry HL >= #C000".

Where an address is returned in bank 0 this is indicated by "address in bank 0".

"BANNED" means the entry cannot be used, this restriction is not enforced – break it at your peril!

Main Firmware Jumpblock

0	KM INITIALISE	OK
1	KM RESET	OK
2	KM WAIT CHAR	OK
3	KM READ CHAR	OK
4	KM CHAR RETURN	OK
5	KM SET EXPAND	Entry HL >= #C000 (but see extended jumpblock)
6	KM GET EXPAND	OK
7	KM EXP BUFFER	Entry DE >= #C000
8	KM WAIT KEY	OK
9	KM READ KEY	OK (but see extended jumpblock)
10	KM TEST KEY	OK
11	KM GET STATE	OK
12	KM GET JOYSTICK	OK
13	KM SET TRANSLATE	OK (but see extended jumpblock)
14	KM GET TRANSLATE	OK
15	KM SET SHIFT	OK (but see extended jumpblock)
16	KM GET SHIFT	OK
17	KM SET CONTROL	OK (but see extended jumpblock)
18	KM GET CONTROL	OK
19	KM SET REPEAT	OK
20	KM GET REPEAT	OK
21	KM SET DELAY	OK (but see extended jumpblock)
22	KM GET DELAY	OK
23	KM ARM BREAKS	BANNED
24	KM DISARM BREAK	OK
25	KM BREAK EVENT	OK
26	TXT INITIALIZE	OK
27	TXT RESET	OK
28	TXT VDU ENABLE	OK
29	TXT VDU DISABLE	OK
30	TXT OUTPUT	OK
31	TXT WR CHAR	OK

32	TXT RD CHAR	OK
33	TXT SET GRAPHIC	OK
34	TXT WIN ENABLE	OK
35	TXT GET WINDOW	OK
36	TXT CLEAR WINDOW	OK
37	TXT SET COLUMN	OK
38	TXT SET ROW	OK
39	TXT SET CURSOR	OK
40	TXT GET CURSOR	OK
41	TXT CUR ENABLE	OK
42	TXT CUR DISABLE	OK
43	TXT CUR ON	OK
44	TXT CUR OFF	OK
45	TXT VALIDATE	OK
46	TXT PLACE CURSOR	OK
47	TXT REMOVE CURSOR	OK
48	TXT SET PEN	OK
49	TXT GET PEN	OK
50	TXT SET PAPER	OK
51	TXT GET PAPER	OK
52	TXT INVERSE	OK
53	TXT SET BACK	OK
54	TXT GET BACK	OK
55	TXT GET MATRIX	Exit HL address in bank 0
56	TXT SET MATRIX	Entry HL >= #C000
57	TXT SET M TABLE	Entry HL >= #C000, Exit HL address in bank 0
58	TXT GET M TABLE	Exit HL address in bank 0
59	TXT GET CONTROLS	Exit HL address in bank 0
60	TXT STR SELECT	OK
61	TXT SWAP STREAMS	OK
62	GRA INITIALISE	OK
63	GRA RESET	OK
64	GRA MOVE ABSOLUTE	OK
65	GRA MOVE RELATIVE	OK
66	GRA ASK CURSOR	OK
67	GRA SET ORIGIN	OK
68	GRA GET ORIGIN	OK
69	GRA WIN WIDTH	OK
70	GRA WIN HEIGHT	OK

71	GRA GET W WIDTH	OK
72	GRA GET W HEIGHT	OK
73	GRA CLEAR WINDOW	OK
74	GRA SET PEN	OK
75	GRA GET PEN	OK
76	GRA SET PAPER	OK
77	GRA GET PAPER	OK
78	GRA PLOT ABSOLUTE	OK
79	GRA PLOT RELATIVE	OK
80	GRA TEST ABSOLUTE	OK
81	GRA TEST RELATIVE	OK
82	GRA LINE ABSOLUTE	OK
83	GRA LINE RELATIVE	OK
84	GRA WR CHAR	OK
85	SCR INITIALISE	BANNED
86	SCR RESET	OK
87	SCR SET OFFSET	OK
88	SCR SET BASE	BANNED
89	SCR GET LOCATION	OK
90	SRC SET MODE	OK (or send Esc 3 mode to the CRT)
91	SCR GET MODE	OK
92	SCR CLEAR	OK
93	SCR CHAR LIMITS	OK
94	SCR CHAR POSITION	Exit HL address in bank 0
95	SCR DOT POSITION	Exit HL address in bank 0
96	SCR NEXT BYTE	Exit HL address in bank 0
97	SCR PREV BYTE	Entry and Exit HL address in bank 0
98	SCR NEXT LINE	Entry and Exit HL address in bank 0
99	SCR PREV LINE	Entry and Exit HL address in bank 0
100	SCR INK ENCODE	OK
101	SCR INK DECODE	OK
102	SCR SET INK	OK (but see extended jumpblock)
103	SCR GET INK	OK
104	SCR SET BORDER	OK (but see extended jumpblock)
105	SCR GET BORDER	OK
106	SCR SET FLASHING	OK (but see extended jumpblock)

107	SCR GET FLASHING	OK
108	SCR FILL BOX	OK
109	SCR FLOOD BOX	Entry HL address in bank 0
110	SCR CHAR INVERT	OK
111	SCR HW ROLL	OK
112	SCR SW ROLL	OK
113	SCR UNPACK	Entry Hl address in bank 0, DE >= #C000
114	SCR REPACK	Entry DE address in bank 0
115	SCR ACCESS	OK
116	SCR PIXELS	Entry HL address in bank 0
117	SCR HORIZONTAL	OK
118	SCR VERTICAL	OK
119	CAS INITIALISE	OK
120	CAS SET SPEED	OK
121	CAS NOISY	OK
122	CAS START MOTOR	OK
123	CAS STOP MOTOR	OK
124	CAS RESTORE MOTOR	OK
125	CAS IN OPEN	Entry DE, HL >= #C000, Exit HL address in bank 0
126	CAS IN CLOSE	OK
127	CAS IN ABANDON	OK
128	CAS IN CHAR	OK
129	CAS IN DIRECT	Entry HL >= #C000
130	CAS RETURN	OK
131	CAS TEST EOF	OK
132	CAS OUT OPEN	Entry DE, HL >= #C000, Exit HL address in bank 0
133	CAS OUT CLOSE	OK
134	CAS OUT ABANDON	OK
135	CAS OUT CHAR	OK
136	CAS OUT DIRECT	Entry HL >= #C000
137	CAS CATALOG	Entry DE >= #C000
138	CAS WRITE	Entry HL >= #C000
139	CAS READ	Entry HL >= #C000
140	CAS CHECK	Entry HL >= #C000
141	SOUND RESET	OK
142	SOUND QUEUE	Entry HL >= #C000

143	SOUND CHECK	OK
144	SOUND ARM EVENT	BANNED
145	SOUND RELEASE	OK
146	SOUND HOLD	OK
147	SOUND CONTINUE	OK
148	SOUND AMPL ENVELOPE	Entry HL >= #C000
149	SOUND TONE ENVELOPE	Entry HL >= #C000
150	SOUND A ADDRESS	Exit HL address in bank 0
151	SOUND T ADDRESS	Exit HL address in bank 0
152	KL CHOKE OFF	BANNED
153	KL ROM WALK	BANNED
154	KL INIT BACK	BANNED
155	KL LOG EXT	BANNED
156	KL FIND COMMAND	BANNED
157	KL NEW FRAME FLY	BANNED
158	KL ADD FRAME FLY	BANNED
159	KL DEL FRAME FLY	BANNED
160	KL NEW FAST TICKER	BANNED
161	KL ADD FAST TICKER	BANNED
162	KL DEL FAST TICKER	BANNED
163	KL ADD TICKER	BANNED
164	KL DEL TICKER	BANNED
165	KL INIT EVENT	BANNED
166	KL EVENT	BANNED
167	KL SYNC RESET	BANNED
168	KL DEL SYNCHRONOUS	BANNED
169	KL NEXT SYNC	BANNED
170	KL DO SYNC	BANNED
171	KL DONE SYNC	BANNED
172	KL EVENT DISABLE	BANNED
173	KL EVENT ENABLE	BANNED
174	KL DISARM EVENT	BANNED
175	KL TIME PLEASE	OK
176	KL TIME SET	OK
177	MC BOOT PROGRAM	BANNED
178	MC START PROGRAM	BANNED
179	MC WAIT FLYBACK	OK
180	MC SET MODE	BANNED
181	MC SCREEN OFFSET	BANNED

182	MC CLEAR INKS	Entry DE >= #C000
183	MC SET INKS	Entry DE >= #C000
184	MC RESET PRINTER	OK
185	MC PRINT CHAR	OK
186	MC BUSY PRINTER	OK
187	MC SEND PRINTER	OK
188	MC SOUND REGISTER	OK
189	JUMP RESTORE	BANNED
190	KM SET LOCKS	OK
191	KM FLUSH	OK
192	TXT ASK STATE	OK
193	GRA DEFAULT	OK
194	GRA SET BACK	OK
195	GRA SET FIRST	OK
196	GRA SET LINE MASK	OK
197	GRA FROM USER	OK
198	GRA FILL	Entry HL >= #C000
199	SCR SET POSITION	BANNED
200	MC PRINT TRANSLATION	Entry HL >= #C000
201	MC BANK SELECT	BANNED

Indirection Jumpblock

BANNED

High Kernel Jumpblock

BANNED

Low Kernel Jumpblock

BANNED

Appendix L

GSX – Virtual Device Interface (VDI) Specification

INTRODUCTION

This appendix contains the specification of the Virtual Device Interface (VDI). The VDI defines how device drivers interface to GDOS, the device-independent portion of GSX. The context for this document is from the DEVICE DRIVER point of view. All coordinate information is assumed to be in device coordinate space.

FORMAT

Function: GSX graphics operation

Input Parameters

contrl(1) –	Opcode for driver function.
contrl(2) –	Number of vertices in array ptsin. Each vertex consists of an x and a y coordinate pair so the length of this array is twice as long as the number of vertices specified.
contrl(4) –	Length of integer array intin.
contrl(6-n) –	Opcode dependent information.
intin –	Array of integer input parameters.
ptsin –	Array of input point coordinate data.

Output Parameters

contrl(3) –	Number of vertices in array ptsout. Each vertex consists of an x and a y coordinate pair so the length of this array is twice as long as the number of vertices specified. Other data may be passed back here depending on the opcode.
contrl(5) –	Length of integer array intout.
contrl(6-n) –	Opcode dependent information.
intout –	Array of integer output point parameters.
ptsout –	Array of output point coordinate data.

Notes All data passed to the device driver is assumed to be 2-byte INTEGERS, including individual characters in character strings.

All coordinates passed to GSX are in Normalized Device Coordinates (0-32767 along each axis). These units are then mapped to the actual device units (for example, rasters for CRTs or steps for plotters and printers) by GSX so that all coordinates passed to the device driver are in device units.

Because both input and output coordinates are converted by GSX, both the calling routine and the device driver must make sure that the input vertex count (contrl(2)) and output vertex count (contrl(3)) are set. The calling routine must set contrl(2) to 0 if no x,y coordinates are are being passed to GSX. Similarly, the device driver must set contrl(3) to 0 if no x,y coordinates are being returned through GSX. Coordinates returned by GSX are assumed to be the bottom left edge of the pixel. As a consequence, points at the top and right edges of the device coordinate system will not be at the edge of the Normalized Device Coordinates (NDC) system. Exactly how far away they will be is device dependent.

Because 0-32767 maps to the full extent on each axis, coordinate values are scaled differently on the x and y axes of devices that do not have a square display.

All references to arrays are 1-based; that is, subscripted element 1 is the first element in the array.

On calls to the GDOS the number of arguments passed in the intin array (contrl(4)), and the maximum size of the intout array (contrl(5)) should be set by the application. On return to the GDOS by the GIOS the number of arguments in the intout array should be set by the GIOS. Refer to Appendixes A and B for GDOS calling conventions for specific operating systems.

All opcodes must be recognized, whether or not they produce any action. If an opcode is out of range then no action is performed. A list of required opcodes for CRT devices, plotters, and printers follows the specification. These opcodes must be present and perform as specified. All opcodes should be implemented whenever possible since full implementation gives better quality graphics.

Device driver I/O (that is, communication between the device driver and the device via the system hardware ports) is done through operating system calls.

OPEN WORKSTATION
Initialize a graphic workstation.

Input	contrl(1) –	Opcode = 1
	contrl(2) –	0
	contrl(4) –	Length of intin = 10
	intin –	Initial defaults (for example, linestyle colour and character size)
	intin(1) –	Workstation identifier (device driver id). This value is used to determine which device driver to dynamically load into memory.
	intin(2) –	Linetype
	intin(3) –	Polyline color index
	intin(4) –	Marker type
	intin(5) –	Polymarker color index
	intin(6) –	Text font
	intin(7) –	Text color index
	intin(8) –	Fill interior style

	intin(9) –	Fill style index
	intin(10) –	Fill color index
Output	contrl(3) –	Number of output vertices = 6
	contrl(5) –	Length of intout = 45
	intout(1) –	Maximum addressable width of screen/plotter in rasters/ steps assuming a 0 start point (for example, a resolution of 640 implies an addressable area of 0-639, so intout(1)=639)
	intout(2) –	Maximum addressable height of screen/plotter in rasters/ steps assuming a 0 start point (for example, a resolution of 480 implies an addressable area of 0-479, so intout(2)=479)
	intout(3) –	Device Coordinate units flag 0 = Device capable of producing precisely scaled image (typically plotters and printers) 1 = Device not capable of precisely scaled image (CRTs)
	intout(4) –	Width of one pixel (plotter step, or aspect ratio for CRT) in micrometers
	intout(5) –	Height of one pixel (plotter step, or aspect ratio for CRT) in micrometers
	intout(6) –	Number of character heights 0 = continuous scaling
	intout(7) –	Number of linetypes
	intout(8) –	Number of line widths
	intout(9) –	Number of marker types
	intout(10) –	Number of marker sizes
	intout(11) –	Number of fonts intout(12) – Number of patterns
	intout(13) –	Number of hatch styles

intout(14) –	Number of predefined colors (must be at least 2 even for monochrome device). This is the number of colors that can be displayed on the device simultaneously.
intout(15) –	Number of Generalized Drawing Primitives (GDPs)
intout(16)-	
intout(25) –	Linear list of GDP numbers supported -1 no more GDPs in list. Application should search list until finding a -1 for the desired GDP. 1 – bar 2 – arc 3 – pie slice 4 – circle 5 – ruling chars
intout(26)-	
intout(35) –	Linear list of attribute set associated with each GDP -1 – no more GDPs 0 – polyline 1 – polymarker 2 – text 3 – fill area 4 – none
intout(36) –	Color capability flag 0 – no 1 – yes
intout(37) –	Text rotation capability flag 0 – no 1 – yes

intout(38) –	Fill area capability flag 0 – no 1 – yes
intout(39) –	Read cell array operation capability flag 0 – no 1 – yes
intout(40) –	Number of available colors (total number of colors in color palette) 0 – continuous device (morethan32767colors) 2 – monochrome (black and white) >2 – number of colors available
intout(41) –	Number of locator devices available
intout(42) –	Number of valuator devices available
intout(43) –	Number of choice devices available
intout(44) –	Number of string devices available
intout(45) –	Workstation type 0 – Output only 1 – Input only 2 – Input/Output 3 – Device independent segment storage 4 – GKS Metafile output
ptsout(1) –	0
ptsout(2) –	Minimum character height in device units (not cell size)
ptsout(3) –	0
ptsout(4) –	Maximum character height in device units (not cell size)

ptsout(5) –	Minimum line width in device units
ptsout(6) –	0
ptsout(7) –	Maximum line width in device units
ptsout(8) –	0
ptsout(9) –	0
ptsout(10) –	Minimum marker height in device units (not cell size)
ptsout(11) –	0
ptsout(12) –	Maximum marker height in device units (not cell size)

The default color table should be set up differently for a monochrome and a color device.

Monochrome CRT type devices

Index	Color
0	Black
1	White

Monochrome Printer/Plotter devices

Index	Color
0	White
1	Black

Color

Index	Color
0	Black
1	Red
2	Green
3	Blue
4	Cyan
5	Yellow 6 Magenta
7	White
8-n	White

Other default values that should be set by the driver during initialization are as follows:

Character height =	Minimum character height
Character up vector =	90 degrees counterclockwise from the right horizontal (0 degrees rotation)
Line width =	1 device unit (raster, plotter step)
Marker height =	Minimum marker height
Writing mode =	Replace
Input mode =	Request for all input classes (locator, valuator, choice, string)

Description The Open Workstation operation causes a graphics device to become the current device for the application program. The device is initialized with the parameters in the input array and information about the device is returned to GDOS. The graphic device is selected, and, if it is a CRT, the screen is cleared and the alpha device is deselected and blanked.

CLOSE WORKSTATION
Stop all graphics output to this workstation.

Input	contrl(1) –	Opcode = 2
	contrl(2) –	0
Output	contrl(3) –	0

Description The Close Workstation operation terminates the graphics device properly and prevents any further output to the device. If the device is a CRT, the alpha device is selected, the screen is cleared, and the graphics device is deselected and blanked. If the device is a printer, then an update is executed.

CLEAR WORKSTATION
Clear CRT screen or prompt for new paper on plotter.

Input	contrl(1) –	Opcode = 3
	contrl(2) –	0
Output	contrl(3) –	0

Description The Clear Workstation operation causes CRT screens to be erased. If the device is a plotter without paper advance, the operator is prompted to load a new page. If the device is a printer a form feed is issued and then an update is executed.

UPDATE WORKSTATION
Display all pending graphics on workstation.

Input	contrl(1) –	Opcode = 4
	contrl(2) –	0
Output	contrl(3) –	0

Description The Update Workstation operation causes all pending graphics commands that are queued to be executed immediately. The operation is analogous to flushing buffers. For printer drivers this call must be used to start output to the printer.

ESCAPE
Perform device specific operation.

Input	contrl(1) –	Opcode = 5
	contrl(2) –	Number of input vertices
	contrl(4) –	Number of input parameters
	contrl(6) –	Function identifier

1=	INQUIRE ADDRESSABLE CHARACTER CELLS
2=	ENTER GRAPHICS MODE
3=	EXIT GRAPHICS MODE
4=	CURSOR UP
5=	CURSOR DOWN
6=	CURSOR RIGHT
7=	CURSOR LEFT
8=	HOME CURSOR
9=	ERASE TO END OF SCREEN
10=	ERASE TO END OF LINE
11=	DIRECT CURSOR ADDRESS
12=	OUTPUT CURSOR ADDRESSABLE TEXT
13=	REVERSE VIDEO ON

14=	REVERSE VIDEO OFF
15=	INQUIRE CURRENT CURSOR ADDRESS
16=	INQUIRE TABLET STATUS
17=	HARDCOPY
18=	PLACE GRAPHIC CURSOR AT LOCATION
19=	REMOVE LAST GRAPHIC CURSOR
20-50=	UNUSED BUT RESERVED FOR FUTURE EXPANSION
51-100=	UNUSED AND AVAILABLE FOR USE

	intin –	Function dependent information (described on following pages)
	ptsin –	Array of input coordinates for escape function
Output	contrl(3) –	Number of output vertices
	contrl(5) –	Number of output parameters
	intout –	Array of output parameters
	ptsout –	Array of output coordinates

Description The Escape operation allows the special capabilities of a graphics device to be accessed from the application program. Some escape functions above are predefined, but others can be defined for your particular devices. The parameters passed are dependent on the function being performed.

ESCAPE: INQUIRE ADDRESSABLE CHARACTER CELLS

Return the number of alpha cursor addressable columns and alpha cursor addressable rows.

Input	contrl(2) –	0
	contrl(6) –	Function ID = 1

Output	contrl(3) –	0
	intout(1) –	Number of addressable rows on the screen, typically 24 (-1 indicates cursor addressing not possible)
	intout(2) –	Number of addressable columns on the screen, typically 80 (-1 indicates cursor addressing not possible)

Description This operation returns information to the calling program about the number of vertical (rows) and horizontal (columns) positions where the alpha cursor can be positioned on the screen.

ESCAPE: ENTER GRAPHICS MODE
Enter graphics mode if different from alpha mode.

Input	contrl(2) –	0
	contrl(6) –	Function id = 2
Output	contrl(3) –	0

Description This operation causes the graphics device to enter the graphics mode if different than the alpha mode. Used to explicitly exit alpha cursor addressing mode and to transition from alpha to graphic mode properly. The graphics device is selected and cleared. The alpha device is deselected and blanked.

ESCAPE: EXIT GRAPHICS MODE
Exit graphics mode if different from alpha mode.

Input	contrl(2) –	0
	contrl(6) –	Function id = 3

Output contrl(3) – 0

Description The Exit Graphics operation causes the graphics device to exit the graphics mode if different than the alpha mode. Used to explicitly enter the alpha cursor addressing mode and to transition from graphics to alpha mode properly. The alpha device is selected and cleared. The graphics device is deselected and blanked.

ESCAPE: CURSOR UP
Move alpha cursor up one row without altering horizontal position.

Input contrl(2) – 0
contrl(6) – Function id = 4

Output contrl(3) – 0

Description This operation moves the alpha cursor up one row without altering the horizontal position. If the cursor is already at the top margin, no action results.

ESCAPE: CURSOR DOWN
Move alpha cursor down one row without altering horizontal position.

Input contrl(2) – 0
contrl(6) – Function id = 5

Output contrl(3) – 0

Description This operation moves the alpha cursor down one row without altering the horizontal position. If the cursor is already at the bottom margin, no action results.

ESCAPE: CURSOR RIGHT
Move alpha cursor right one column without altering vertical position.

Input	contrl(2) –	0
	contrl(6) –	Function id = 6
Output	contrl(3) –	0

Description The Cursor Right operation moves the alpha cursor right one column without altering the vertical position. If the cursor is already at the right margin, no action results

ESCAPE: CURSOR LEFT
Move alpha cursor left one column without altering vertical position.

Input	contrl(2) –	0
	contrl(6) –	Function id = 7
Output	contrl(3) –	0

Description The Cursor Left operation causes the alpha cursor to move one column to the left without altering the vertical position. If the cursor is already at the left margin, no action results.

ESCAPE: HOME CURSOR
Send cursor to home position.

Input	contrl(2) –	0
	contrl(6) –	Function id = 8
Output	contrl(3) –	0

Description This operation causes the alpha cursor to move to the home position, usually the upper left corner of a CRT display.

ESCAPE: ERASE TO END OF SCREEN
Erase from current alpha cursor position to the end of the screen.

Input	contrl(2) –	0
	contrl(6) –	Function id = 9
Output	contrl(3) –	0

Description This operation erases the display surface from the current alpha cursor position to the end of the screen. The current alpha cursor location does not change.

ESCAPE: ERASE TO END OF LINE
Erase from the current alpha cursor position to the end of the line.

Input	contrl(2) –	0
	contrl(6) –	Function id = 10
Output	contrl(3) –	0

Description This operation erases the display surface from the current alpha cursor position to the end of the current line. The current alpha cursor location does not change.

ESCAPE: DIRECT CURSOR ADDRESS
Move alpha cursor to specified row and column.

Input	contrl(2) –	0
	contrl(6) –	Function id = 11
	intin(1) –	Row number (1 - number of rows)
	intin(2) –	Column number (1 - number of columns)

Output	contrl(3) –	0

Description The Direct Cursor Address operation moves the alpha cursor directly to the specified row and column address anywhere on the display surface. Addresses that are beyond the range that can be displayed on the screen are set to the maximum row and/or column accordingly.

ESCAPE: OUTPUT CURSOR ADDRESSABLE TEXT
Output text at the current alpha cursor position.

Input	contrl(2) –	0
	contrl(4) –	Number of characters in character **string**
	contrl(6) –	Function id = 12
	intin –	Text string in ASCII
Output	contrl(3) –	0

Description This operation displays a string of text starting at the current cursor position. Alpha text characteristics are determined by the attributes currently in effect (for example, reverse video).

ESCAPE: REVERSE VIDEO ON
Display subsequent cursor addressable text in reverse video.

Input	contrl(2) –	0
	contrl(6) –	Function id = 13
Output	contrl(3) –	0

Description This operation causes all subsequent text to be displayed in reverse video format; that is, characters are dark on a light background.

ESCAPE: REVERSE VIDEO OFF

Display subsequent cursor addressable text in standard video.

Input	contrl(2) –	0
	contrl(6) –	Function id = 14
Output	contrl(3) –	0

Description This operation causes all subsequent text to be displayed in normal video format; that is, characters are light on a dark background.

ESCAPE: INQUIRE CURRENT CURSOR ADDRESS

Return the current cursor position.

Input	contrl(2) –	0
	contrl(6) –	Function id = 15
Output	contrl(3) –	0
	intout(1) –	Row number (1 - number of rows)
	intout(2) –	Column number (1 - number of columns

Description This operation returns the current position of the alpha cursor in row, column coordinates.

ESCAPE: INQUIRE TABLET STATUS

Return tablet status.

Input	contrl(2) –	0
	contrl(6) –	Function id = 16
Output	contrl(3) –	0
	intout(1) –	tablet status
		0 = tablet not available
		1 = tablet available

Description This operation returns tablet status whether a graphics tablet, mouse, joystick, or other similar devices are connected to the workstation.

ESCAPE: HARD COPY

Generate hardcopy.

Input	contrl(2) –	0
	contrl(6) –	Function id = 17
Output	contrl(3) –	0

Description	This operation causes the device to generate a hardcopy. This function is very device specific and can entail copying the screen to a printer or other attached hardcopy device.

ESCAPE: PLACE GRAPHIC CURSOR AT LOCATION

Place a graphic cursor at specified location

Input	contrl(2) –	2
	contrl(6) –	Function id = 18
	ptsin(1) –	x-coordinate of location to place cursor
	ptsin(2) –	y-coordinate of location to place cursor
Output	contrl(3) –	0

Description Place Graphic Cursor at the specified location. This is device dependent and can be an underbar, block, or similar character. This cursor should be the same type as used for request mode locator input. In this way, if sample mode input is supported, the application may use this call to generate the cursor for rubber band type drawing. In memory mapped devices, it is drawn in XOR mode so that it can be removed. The cursor has no attributes; for example, style or colour index.

ESCAPE: REMOVE LAST GRAPHIC CURSOR

Remove last graphic cursor/marker.

Input	contrl(2) –	0
	contrl(6) –	Function id = 19
Output	contrl(3) –	0

Description This operation removes the last graphic cursor placed on the screen.

POLYLINE
Output a polyine to device.

Input	contrl(1) –	Opcode = 6
	contrl(2) –	Number of vertices (x,y pairs) in polyline (n)
	ptsin –	Array of coordinates of polyline in device units (for example, rasters and plotter steps)
	ptsin(1) –	x-coordinate of first point
	ptsin(2) –	y-coordinate of first point
	ptsin(3) –	x-coordinate of second point
	ptsin(4) –	y-coordinate of second point
	.	
	.	
	ptsin(2n-1) –	x-coordinate of last point
	ptsin(2n) –	y-coordinate of last point
Output	contrl(3) –	0

Description This operation causes a polyline to be displayed on the graphics device. The starting point for the polyline is the first point in the input array. Lines are drawn between subsequent points in the array. Make sure that the lines exhibit the current line attributes: colour, linetype, line width. 0 length lines should be displayed. A single coordinate pair should not be displayed.

POLYMARKER
Output markers to the device.

Input	contrl(1) –	Opcode = 7
	contrl(2) –	Number of markers
	ptsin –	Array of coordinates in device units (n) (for example, rasters and plotter steps)
	ptsin(1) –	x-coordinate of first marker
	ptsin(2) –	y-coordinate of first marker

	ptsin(3) –	x-coordinate of second marker
	ptsin(4) –	y-coordinate of second marker
	.	
	.	
	ptsin(2n-1) –	x-coordinate of last marker
	ptsin(2n) –	y-coordinate of last marker
Output	contrl(3) –	0

Description This operation causes markers to be drawn at the points specified in the input array. Make sure the markers display the current attributes: colour, scale, and type.

TEXT
Write text at specified position.

Input	contrl(1) –	Opcode = 8
	contrl(2) –	Number of vertices = 1
	contrl(4) –	Number of characters in text string
	intin –	Word character string in ASCII units
	ptsin(2) –	y-coordinate of start point of text in device units
Output	contrl(3) –	0

Description This operation writes text to the display surface starting at the position specified by the input parameters. Note that the X,Y position specified is the lower left corner of the character itself, not the character cell. Also, make sure the text exhibits current text attributes: colour, height, character up vector, font. Each word of the intin array contains only one character. Any character code out of range for the selected font should be mapped to a blank.

FILLED AREA
Fill a polygon.

Input	contrl(1) –	Opcode = 9
	contrl(2) –	Number of vertices in polygon (n)

	ptsin – Array of coordinates of polygon in device units	
	ptsin(1) –	x-coordinate of first point
	ptsin(2) –	y-coordinate of first point
	ptsin(3) –	x-coordinate of second point
	ptsin(4) –	y-coordinate of second point
	.	
	.	
	ptsin(2n-1) –	x-coordinate of last point
	ptsin(2n) –	y-coordinate of last point
Output	contrl(3) –	0

Description This operation fills a polygon specified by the input array with the current fill colour. Ensure the correct colour, fill interior style (hollow, solid, pattern or hatch) and fill style index are in effect before doing the fill.

If the device cannot do area fill, it must at least outline the polygon in the current fill colour. The device driver must ensure that the fill area is closed by connecting the first point to the last point.

A polygon with zero area should be displayed as a dot. A polygon with only one endpoint should not be displayed.

CELL ARRAY
Display cell array.

Input	contrl(1) –	Opcode = 10
	contrl(2) –	2
	contrl(4) –	Length of colour index array
	contrl(6) –	Length of each row in colour index array (size as declared in a high level language)
	contrl(7) –	Number of elements used in each row of colour index array
	contrl(8) –	Number of rows in colour index array

contrl(9) –	Pixel operation to be performed	
		1 – replace
		2 – overstrike
		3 – complement (xor)
		4 – erase
	intin(1) –	Colour index array (stored one row at time)
	ptsin(1) –	x-coordinate of lower left corner in device units
	ptsin(2) –	y-coordinate of lower left corner in device units
	ptsin(3) –	x-coordinate of upper right corner in device units
	ptsin(4) –	y-coordinate of upper right corner in device units
Output	contrl(3) –	0

Description The Cell Array operation causes the device to draw a rectangular array which is defined by the input parameter X,Y coordinates and the colour index array.

The extents of the cell are defined by the lower left-hand and the upper right-hand X,Y coordinates. Within the rectangle defined by those points, the colour index array specifies colours for individual components of the cell.

Each row of the colour index array should be expanded to fill the entire width of the rectangle specified if necessary, via pixel replication. Each row of the colour index array should also be replicated the appropriate number of times to fill the entire height of the rectangular area.

If the device cannot do cell arrays it must at least outline the area in the current line colour.

GENERALIZED DRAWING PRIMITIVE (GDP)
Output a primitive display element.

Input	contrl(1) –	Opcode = 11
	contrl(2) –	Number of vertices in ptsin
	contrl(4) –	Length of input array intin
	contrl(6) –	Primitive id

1 – BAR – uses fill area attributes (interior style, fill style, fill colour)
2 – ARC – uses line attributes (colour, linetype, width)
3 – PIE SLICE – uses fill area attributes (interior style, fill style, fill colour)
4 – CIRCLE – uses fill area attributes (interior style, fill style, fill colour)
5 – PRINT GRAPHIC CHARAC-TERS (RULING CHARACT-ERS)
6 – 7 are unused but reserved for future expansion
8 – 10 are unused and available for use

ptsin –	Array of coordinates for GDP
ptsin(1) –	x-coordinate of first point
ptsin(2) –	y-coordinate of first point
ptsin(3) –	x-coordinate of second point
ptsin(4) –	y-coordinate of second point
.	
.	
ptsin(2n-1) –	x-coordinate of last point
ptsin(2n) –	y-coordinate of last point
intin –	Data record

BAR

contrl(2) –	2 (number of vertices)
contrl(6) –	1 (primitive ID)
ptsin(1) –	x-coordinate of lower left-hand corner of bar
ptsin(2) –	y-coordinate of lower left-hand corner of bar
ptsin(3) –	x-coordinate of upper right-hand corner of bar
ptsin(4) –	y-coordinate of upper right-hand corner of bar

ARC AND PIE SLICE

contrl(2) –	4 (number of vertices)
contrl(6) –	2 (ARC) or 3 (PIE SLICE)
intin(1) –	Start angle in tenths of degrees (0-3600)
intin(2) –	End angle in tenths of degrees (0-3600)
ptsin(1) –	x-coordinate of center point of arc
ptsin(2) –	y-coordinate of center point of arc
ptsin(3) –	x-coordinate of start point of arc on circumference
ptsin(4) –	y-coordinate of start point of arc on circumference
ptsin(5) –	x-coordinate of end point of arc on circumference
ptsin(6) –	y-coordinate of end point of arc on circumference
ptsin(7) –	Radius
ptsin(8) –	0

CIRCLE

contrl(2) –	3 (number of points)
contrl(6) –	4 (primitive id)
ptsin(1) –	x-coordinate of center point of circle
ptsin(2) –	y-coordinate of center point of circle
ptsin(3) –	x-coordinate of point on circumference
ptsin(4) –	y-coordinate of point on circumference
ptsin(5) –	Radius
ptsin(6) –	0

PRINT GRAPHIC CHARACTER

For graphics on printer (such as Diablo and Epson)

contrl(2) –	1 (number of points)
contrl(4) –	Number of characters to output
contrl(6) –	5
intin –	Graphic characters to output
ptsin(1) –	x-coordinate of start point of characters
ptsin(2) –	y-coordinate of start point of characters

Output	contrl(3) – 0

Description The Generalized Drawing Primitive (GDP) operation allows you to take advantage of the intrinsic drawing capabilities of your graphics device. Special elements such as arcs and circles can be accessed through this mechanism. Several primitive identifiers are predefined and others are available for expansion.

The control and data arrays are dependent on the nature of the primitive.

In some GDPs (Arc, Circle, Pie slice) redundant but consistent information is provided. Only the necessary information for a particular device need be used. Also, all angle specifications assume that 0 degrees is 90 degrees to the right of vertical, with values increasing in the counterclockwise direction.

SET CHARACTER HEIGHT
Set character height.

Input	contrl(1) –	Opcode = 12
	contrl(2) –	Number of vertices = 1
	ptsin(1) –	0
	ptsin(2) –	Requested character height in device units (rasters, plotter steps)
Output	contrl(3) –	Number of vertices = 2
	ptsout(1) –	Actual character width selected in device units
	ptsout(2) –	Actual character height selected in device units
	ptsout(3) –	Character cell width in device units
	ptsout(4) –	Character cell height in device units

Description This operation sets the current text character height in Device Units. The specified height is the height of the character itself rather than the character cell. The driver returns the size of both the character and the character cell. The character size is defined as the size of an uppercase W. If the requested size does not exist, a smaller size should be used.

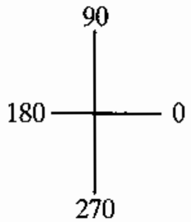
```
            10000010
            10000010
            10000010
            10010010    CHARACTER  CELL
            10101010    HEIGHT     HEIGHT
            11000110
ORIGIN OF   10000010    BASE LINE
ROTATION    00000000

            10000010
            10000010
            10000010
            10010010    CHARACTER  CELL
            10101010    HEIGHT     HEIGHT
            11000110
ORIGIN OF   10000010    BASE LINE
ROTATION    00000000
```

SET CHARACTER UP VECTOR
Set text direction.

Input	contrl(1) –	Opcode = 13
	contrl(2) –	0
	intin(1) –	Requested angle of rotation of character baseline (in tenths of degrees 0 - 3600)
	intin(2) –	Run of angle = cos (angle) * 100 (0-100)
	intin(3) –	Rise of angle = sin (angle) * 100 (0-100)
Output	contrl(3) –	0
	contrl(5) –	1
	intout(1) –	Angle of rotation of character baseline selected (in tenths of degrees 0-3600)

Description This operation requests an angle of rotation specified in tenths of degrees for the CHARACTER UP VECTOR, which specifies the baseline for subsequent text. The driver returns the actual up direction that is a best fit match to the requested value.

For convenience, redundant but consistent information is provided on input. Only information pertinent to a given device need be used. The angle specification assumes that 0 degrees is 90 degrees to the right of vertical (east on a compass), with angles increasing in the counterclockwise direction.



SET COLOUR REPRESENTATION
Specify colour index value.

Input	contrl(1) –	Opcode = 14
	contrl(2) –	0
	intin(1) –	Colour index
	intin(2) –	Red colour intensity (in tenths of percent 0-1000)
	intin(3) –	Green colour intensity
	intin(4) –	Blue colour intensity
Output	contrl(3) –	0

Description This operation associates a colour index with the colour specified in RGB units. At least two colour indexes are required (black and white for monochrome). On a monochrome device, any percentage of colour should be mapped to white. On colour devices without palettes, a simple remapping of the colour indexes is sufficient. On colour devices with palettes, loading the palette map is the proper operation. If the colour index requested is out of range, no operation is performed.

SET POLYLINE LINETYPE
Set polyline linetype.

Input	contrl(1) –	Opcode = 15
	contrl(2) –	0
	intin(1) –	Requested linestyle
Output	contrl(3) –	0
	intout(1) –	Linestyle selected

Description This operation sets the linetype for subsequent polyline operations. The total number of linestyles available is device dependent; however, 5 linestyles are required: one solid plus four dash styles.

If the requested linestyle is out of range, use linestyle 1 (solid).

STYLE –	1 SOLID	1111111111111111
STYLE –	2 DASH	1111111000000000
STYLE –	3 DOT	1110000011100000
STYLE –	4 DASH,DOT	1111111000111000
STYLE –	5 LONG DASH	1111111111110000

SET POLYLINE LINE WIDTH
Set polyline line width.

Input	contrl(1) –	Opcode = 16
	contrl(2) –	Number of input vertices = 1
	ptsin(1) –	Requested line width in device units
	ptsin(2) –	0
Output	contrl(3) –	Number of output vertices = 1
	ptsout(1) –	Selected line width in device units
	ptsout(2) –	0

Description This operation sets the width of lines for subsequent polyline operations. Any attempt to set the width beyond the specified maximum will set it to the maximum line width.

SET POLYLINE Colour INDEX
Set polyline colour index.

Input	contrl(1) –	Opcode = 17
	contrl(2) –	0
	intin(1) –	Requested colour index
Output	contrl(3) –	0
	intout(1) –	Colour index selected

Description	This operation sets the colour index for subsequent polyline operations. The colour signified by the index is determined by the SET–Colour REPRES-ENTATION operation. At least two colour indexes are required. Colour indexes range from 0 to a device-dependent maximum. If the selected index is out of range, use the MAXIMUM colour index.

SET POLYMARKER TYPE
Set polymarker type.

Input	contrl(1) –	Opcode = 18
	contrl(2) –	0
	intin(1) –	Requested polymarker type
Output	contrl(3) –	0
	intout(1) –	Polymarker type selected

Description This operation sets the marker type for subsequent polymarker operations. The total number of markers available is device-dependent; however, five marker types are required, as follows:

1 -	.	Dot
2 -	+	Plus
3 -	*	Asterisk
4 -	O	Circle
5 -	X	Diagonal Cross

If the requested marker type is out of range, use type 3. Marker 1 should always be implemented as the smallest dot that can be displayed.

SET POLYMARKER SCALE
Set polymarker scale (height).

Input	contrl(1) –	Opcode = 19
	contrl(2) –	Number of input vertices = 1

	ptsin(1) –	0
	ptsin(2) –	Requested polymarker height in device units
Output	contrl(3) –	Number of output vertices = 1
	ptsout(1) –	0
	ptsout(2) –	Polymarker height selected in device units

Description This operation requests a polymarker height for subsequent polymarker operations. The driver returns the actual height selected. If the selected height does not exist, use a smaller height.

SET POLYMARKER Colour INDEX
Set polymarker colour index.

Input	contrl(1) –	Opcode = 20
	contrl(2) –	0
	intin(1) –	Requested polymarker colour index
Output	contrl(3) –	0
	intout(1) –	Polymarker colour index selected

Description This operation sets the colour index for subsequent polymarker operations. The value of the index is specified by the Colour operation. At least two colour indexes are required. If the index is out of range, use the MAXIMUM colour index.

SET TEXT FONT
Set the hardware text font.

Input	contrl(1) –	Opcode = 21
	contrl(2) –	0
	intin(1) –	Requested hardware text font number

Output	contrl(3) –	0
	intout(1) –	Hardware text font selected

Description This operation selects a character font for subsequent text operations. Fonts are device-dependent and are specified from 1 to a device-dependent maximum.

SET TEXT Colour INDEX
Set colour index.

Input	contrl(1) –	Opcode = 22
	contrl(2) –	0
	intin(1) –	Requested text colour index
Output	contrl(3) –	0
	intout(1) –	Text colour index selected

Description This operation sets the colour index for subsequent text operations. At least two colour indexes are required. Colour indexes range from 0 to a device-dependent maximum. If the selected index is out of range, use the MAXIMUM index.

SET FILL INTERIOR STYLE
Set interior fill style.

Input	contrl(1) –	Opcode = 23
	contrl(2) –	0
	intin(1) –	Requested fill interior style

0 - Hollow (outline no fill)
1 - Solid
2 - Halftone pattern
3 - Hatch

Output	contrl(3) –	0
	intout(1) –	Fill interior style selected

Description	This operation sets the fill interior style to be used in subsequent polygon fill operations. If the requested style is not available, use Hollow. The style actually used is returned to the calling program.

SET FILL STYLE INDEX
Set fill style index.

Input	contrl(1) –	Opcode = 24
	contrl(2) –	0
	intin(1) –	Requested fill style index for Pattern or Hatch fill
Output	contrl(3) –	0
	intout(1) –	fill style index selected for Pattern or Hatch fill

Description Select a fill style based on the fill interior style. This index has no effect if the interior style is either Hollow or Solid. Indexes go from 1 to a device-dependent maximum. If the requested index is not available, use index 1. The index references a hatch style if the fill interior style is hatch, or it references a halftone pattern if the interior fill style is halftone pattern. For consistency, the hatch styles should be implemented in the following order:

1 – vertical lines
2 – horizontal lines
3 – +45o lines
4 – -45o lines
5 – cross
6 – X
>6 – device-dependent

You can implement halftone patterns for gray scale shading with values 1 through 6. Value 1 is the lightest, and 6 is the darkest.

SET FILL Colour **INDEX**
Set fill colour index.

Input	contrl(1) –	Opcode = 25
	contrl(2) –	0
	intin(1) –	Requested fill colour index
Output	contrl(3) –	0
	intout(1) –	Fill colour index selected

Description This operation sets the colour index for subsequent polygon fill operations. The actual RGB value of the colour index is determined by the SET-Colour-REPRESENTATION operation. At least two colour indexes are required. Colour indexes range from 0 to a device-dependent maximum. If the selected index is out of range, use the MAXIMUM.

INQUIRE Colour **REPRESENTATION**
Return colour representation.

Input	contrl(1) –	Opcode = 26
	contrl(2) –	0
	intin(1) –	Requested colour index
	intin(2) –	Set or realized flag 0 = set (return colour values requested) 1 = realized (return colour values realized on device)
Output	contrl(3) –	0
	intout(1) –	Colour index
	intout(2) –	Red intensity (in tenths of percent 0-1000)
	intout(3) –	Green intensity
	intout(4) –	Blue intensity

Description	This operation returns the requested or the actual value of the specified colour index in RGB units. **Note:** The device driver must maintain tables of the colour values that were set (requested) and the colour values that were realized. On devices that have a continuous colour range, one of these tables may not be necessary. If the selected index is out of range, use the values for the MAXIMUM colour index.

INQUIRE CELL ARRAY
Return cell array definition.

Input	contrl(1) –	Opcode = 27
	contrl(2) –	2
	contrl(4) –	Length of colour index array
	contrl(6) –	Length of each row in colour index array
	contrl(7) –	Number of rows in colour index array
	ptsin(1) –	x-coordinate of lower left corner in device units
	ptsin(2) –	y-coordinate of lower left corner in device units
	ptsin(3) –	x-coordinate of upper right corner in device units
	ptsin(4) –	y-coordinate of upper right corner in device units
Output	contrl(3) –	0
	contrl(8) –	Number of elements used in each row of colour index array
	contrl(9) –	Number of rows used in colour index array
	contrl(10) –	Invalid value flag 0 – If no errors 1 – If a colour value could not be determined for some pixel

intout –	Colour index array (stored one row at time) -1 Indicates that a colour index could not be determined for that particular pixel

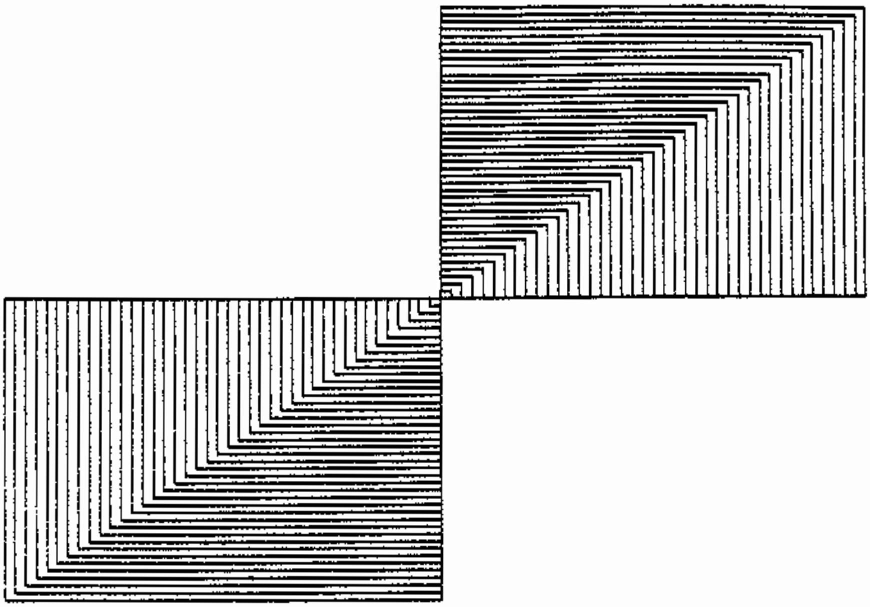
Description This operation returns the cell array definition of the specified cell. Colour indexes are returned one row at a time, starting from the top of the rectangular area, proceeding downward.

Example program to demonstrate use of GSX

```
100 REM *** REM statements are for clarity, THEY ARE NOT USED by the
110 REM *** program so can be left out when entering listings
120 :
130 OPTION BASE 1
140 GOSUB 450           ; REM Set up machine code to pass parameters
150 GOSUB 320           ; REM Reset Workstation, Enter graphics mode
160 :
170 contrl%(1)=11       ; REM Generalised Drawing Primitive
180 contrl%(2)=2        ; REM Number of vertices in ptsin%
190 contrl%(6)=1        ; REM Length of input array intin%
200 ptsin%(1,1)=16384   ; REM X coordinate of first pair (centre)
210 ptsin%(2,1)=16384   ; REM Y coordinate "    "     "  (  "   )
220 :
230 FOR i=1 TO 32767 STEP 400  ; REM 32767=Size of screen in GSX units
240  ptsin%(1,2)=i      ; REM X coordinate of second pair
250  ptsin%(2,2)=i      ; REM Y coordinate "    "     "
260  GOSUB 560          ; REM Call GSX
270 NEXT i
280 :
290 PRINT CHR$(27);"e" ; REM Bring back cursor
300 END
310 :
320 contrl%(1)=1        ; REM Open Workstation
330 contrl%(2)=0        ; REM Always zero
340 contrl%(4)=10       ; REM Length of intin% (always 10 parameters required)
350 intin%(1)=1         ; REM Device driver loaded to use GSX
360                     ; REM Device 1=screen
```

```
370 FOR i=2 TO 10
380  intin%(i)=1          ; REM Only need to be set to 1 because the GSX driver
390                       ; REM supplied with Amstrad's isn't a full implementation
400                       ; REM the full version is supplied with DR Draw and Graph
410 NEXT i
420 GOSUB 560             ; REM Call GSX
430 RETURN
440 :
450 MEMORY HIMEM-7        ; REM Only 7 bytes required, but need to be protected
460 :
470 DATA &h50,&h59,&h0e,&h73,&hc3,&h05,&h00  ; REM Code to pass values to GSX
480 :
490 FOR i=1 TO 7:READ a:POKE HIMEM+i,a:NEXT i: REM Store Machine code
500 :
510 DIM contrl%(6),intin%(80),intout%(45),ptsin%(2,74),ptsout%(2,74)
520 :
530 gsx%=UNT(HIMEM+1)  ; REM Point to GSX calling program
540 RETURN
550 :
560 CALL gsx%(contrl%(1),contrl%(1),contrl%(1),intin%(1),ptsin%(1,1),intout%(1),
570 RETURN                                                              ptsout%(1,1))
```

Output produced from program



Glossary

ambiguous filename: Filename that contains either of the CP/M Plus wildcard characters ? or * in the primary filename or the filetype or both. When you use wildcard characters you create an ambiguous filespec and can easily reference more than one CP/M Plus file. See Section 2 of this manual.

applications program: Program that solves a specific problem. Typical applications programs are business accounting packages word processing (editing) programs and mailing list programs.

argument: Symbol indicating a place into which you can substitute a number letter or name to give an appropriate meaning to a command line.

ASCII: The American Standard Code for Information Interchange is a standard code for representation of numbers letters and symbols. An ASCII text file is a file that can be intelligibly displayed on the video screen or printed on paper. See Appendix B.

attribute: File characteristic that can be set to on or off.

back-up: Copy of a disk or file made for safe keeping or the creation of the back-up disk or file.

bit: Switch in memory that can be set to on (1) or off (0). Bits are grouped into bytes.

block: Area of disk.

bootstrap: Process of loading an operating system into memory. Bootstrap procedures vary from system to system. The boot for an operating system must be customized for the memory size and hardware environment that the operating system manages. Typically the boot is loaded automatically and executed at power up or when the computer is reset. Sometimes called a "cold start."

buffer: Area of memory that temporarily stores data during the transfer of information.

built-in commands: Commands that permanently reside in memory. They respond quickly because they are not accessed from a disk.

byte: Unit of memory or disk storage containing eight bits.

character string: Any combination of letters numbers or special characters on your keyboard. space

command: Elements of a CP/M Plus command line. In general a CP/M Plus command has three parts: the command keyword the command tail and a carriage return keystroke.

command file: Series of coded machine executable instructions stored on disk as a program file invoked in CP/M Plus by typing the command keyword next to the system prompt on the console. CP/M Plus command files generally have a filetype of COM. Files are either command files or data files. Same as a command program.

command keyword: Name that identifies a CP/M Plus command usually the primary filename of a file of type COM or a built-in command. The command keyword precedes the command tail and the carriage return in the command line.

command syntax: Statement that defines the correct way to enter a command. The correct structure generally includes the command keyword the command tail and a carriage return. A syntax line usually contains symbols that you should replace with actual values when you enter the command.

command tail: Part of a command that follows the command keyword in the command line. The command tail can include a drive specification a filename and/or filetype and options or parameters but cannot exceed 128 characters. Some commands do not require a command tail.

concatenate: Term that describes one of PIP's operations that combines two or more separate files into one new file in the specified sequence.

console: Primary input/output device. The console consists of a listing device such as a screen and a keyboard through which the user communicates with the operating system or applications program.

control character: Nonprinting character combination that sends a simple command to CP/M Plus. Some control characters perform line editing functions. To enter a control character hold down the CTRL key on your terminal and strike the character key specified. See Appendix D.

cursor: One-character symbol that can appear anywhere on the console screen. The cursor indicates the position where the next keystroke at the console will have an effect.

data file: Nonexecutable collection of similar information that generally requires a command file to manipulate it.

default: Currently selected disk drive and/or user number. Any command that does not specify a disk drive or a user number references the default disk drive and user number. When CP/M Plus is first invoked the default disk drive is drive A and the default user number is 0 until changed with the USER command.

delimiter: Special characters that separate different items in a command line. For example in CP/M Plus a colon separates the drive spec from the filename. A period separates the filename from the filetype. Brackets separate any options from their command or filespec. Commas separate one item in an option list from another. All of the preceding special characters are delimiters.

directory: Portion of a disk that contains descriptions of each file on the disk. In response to the DIR command CP/M Plus displays the filenames stored in the directory.

DIR attribute: File attribute. A file with the DIR attribute can be displayed by a DIR command. The file can be accessed from the default user number only.

disk diskette: Magnetic media used to store information. Programs and data are recorded on the disk in the same way that music is recorded on a cassette tape. The term diskette refers to smaller capacity removable floppy diskettes. Disk can refer to a diskette a removable cartridge disk or a fixed hard disk.

disk drive: Peripheral device that reads and writes on hard or floppy disks. CP/M Plus assigns a letter to each drive under its control. For example CP/M Plus can refer to the drives in a four-drive system as A B C and D.

editor: Utility program that creates and modifies text files. An editor can be used for creation of documents or creation of code for computer programs. The CP/M Plus editor is invoked by typing the command ED next to the system prompt on the console. (See ED in Section 6 of this manual).

executable: Ready to be run by the computer. Executable code is a series of instructions that can be carried out by the computer. For example the computer cannot execute names and addresses but it can execute a program that prints all those names and addresses on mailing labels.

execute a program: Start a program executing. When a program is running the computer is executing a sequence of instructions.

FCB: See **File Control Block.**

file: Collection of characters instructions or data stored on a disk. The user can create files on a disk.

File Control Block: Structure used for accessing files on disk. Contains the drive filename filetype and other information describing a file to be accessed or created on the disk.

filename: Name assigned to a file. A filename can include a primary filename of 1-8 characters and a filetype of 0-3 characters. A period separates the primary filename from the filetype.

file specification: Unique file identifier. A complete CP/M Plus file specification includes a disk drive specification followed by a colon (d:) a primary filename of 1 to 8 characters a period and a filetype of 0 to 3 characters. For example b:example.tex is a complete CP/M Plus file specification.

filetype: Extension to a filename. A filetype can be from 0 to 3 characters and must be separated from the primary filename by a period. A filetype can tell something about the file. Certain programs require that files to be processed have certain filetypes (see Appendix C).

floppy disk: Flexible magnetic disk used to store information. Floppy disks come in 5 1/4- and 8-inch diameters.

hard disk: Rigid platter-like magnetic disk sealed in a container. A hard disk stores more information than a floppy disk.

hardware: Physical components of a computer.

hex file: ASCII-printable representation of a command (machine language) file.

hexadecimal notation: Notation for the base 16 number system using the symbols 0 1 2 3 4 5 6 7 8 9 A B C D E and F to represent the sixteen digits. Machine code is often converted to hexadecimal notation because it can be easily represented by ASCII characters and therefore printed on the console screen or on paper (see Appendix B).

input: Data going into the system usually from an operator typing at the terminal or by a program reading from the disk.

interface: Object that allows two independent systems to communicate with each other as an interface between hardware and software in a microcomputer.

I/O: Abbreviation for input/output.

keyword: See **command keyword.**

kilobyte: 1024 bytes denoted as 1K. 32 kilobytes equal 32K. 1024 kilobytes equal one megabyte or over one million bytes.

list device: Device such as a printer onto which data can be listed or printed.

logical: Representation of something that might or might not be the same in its actual physical form. For example a hard disk can occupy one physical drive and yet you can divide the available storage on it to appear to the user as if it were in several different drives. These apparent drives are the logical drives.

megabyte: Over one million bytes; 1024 kilobytes. See **byte** and **kilobyte**.

microprocessor: Silicon chip that is the Central Processing Unit (CPU) of the microcomputer.

operating system: Collection of programs that supervises the running of other programs and the management of computer resources. An operating system provides an orderly input/output environment between the computer and its peripheral devices.

option: One of many parameters that can be part of a command tail. Use options to specifiy additional conditions for a command's execution.

output: Data that the system sends to the console or disk.

parameter: Value in the command tail that provides additional information for the command. Technically a parameter is a required element of a command.

peripheral devices: Devices external to the CPU. For example terminals printers and disk drives are common peripheral devices that are not part of the processor but are used in conjunction with it.

physical environment: Actual hardware of a computer. The physical environment varies from computer to computer.

primary filename: First 8 characters of a filename. The primary filename is a unique name that helps the user identify the file contents. A primary filename contains 1 to 8 characters and can include any letter or number and some special characters. The primary filename follows the optional drive specification and precedes the optional filetype.

program: Series of specially coded instructions that performs specific tasks when executed by a computer.

prompt: Characters displayed on the screen to help the user decide what the next appropriate action is. A system prompt is a special prompt displayed by the operating system. The system prompt indicates to the user that the operating system is ready to accept input. The CP/M Plus system prompt is an alphabetic character followed by an angle bracket. The alphabetic character indicates the default drive. Some applications programs have their own special system prompts.

Read-Only: Attribute that can be assigned to a disk file or a disk drive. When assigned to a file the Read-Only attribute allows you to read from that file but not change it. When assigned to a drive the Read-Only attribute allows you to read any file on the disk but prevents you from adding a new file erasing or changing a file renaming a file or writing on the disk. The SET command can set a file or a drive to Read-Only. Every file and drive is either Read-Only or Read-Write. The default setting for drives and files is Read-Write but an error in resetting the disk or changing media automatically sets the drive to Read-Only until the error is corrected. Files and disk drives can be set to either Read-Only or Read-Write.

Read-Write: Attribute that can be assigned to a disk file or a disk drive. The Read-Write attribute allows you to read from and write to a specific Read-Write file or to any file on a disk that is in a drive set to Read-Write. A file or drive can be set to either Read-Only or Read-Write.

record: Collection of data. A file consists of one or more records stored on disk. A CP/M Plus record is 128 bytes long.

RO: See **Read-Only.**

RW: See **Read-Write.**

sector: Portion of a disk track. There are a specified number of sectors on each track.

software: Specially coded programs that transmit machine-readable instructions to the computer as opposed to hardware which is the actual physical components of a computer.

source file: ASCII text file that is an input file for a processing program such as an editor text formatter or assembler.

string: See **character string**

syntax: Format for entering a given command.

system attribute: File attribute. You can give a file the system attribute by using the SYS option in the SET command. A file with the SYS attribute is not displayed in response to a DIR command; you must use DIRS (see Section 5). If you give a file with user number 0 the SYS attribute you can read and execute that file from any user number on the same drive. Use this feature to make your commonly used programs available under any user number.

system prompt: Symbol displayed by the operating system indicating that the system is ready to receive input. See **prompt.**

terminal: See **console.**

track: Concentric rings dividing a disk.

turn-key application: Application designed for the noncomputer-oriented user. For example a typical turn-key application is designed so that the operator needs only to turn on the computer insert the proper program disk and select the desired procedure from a selection of functions (menu) displayed on the screen.

upward-compatible: Term meaning that a program created for the previously released operating system (or compiler etc.) runs under the newly released version of the same operating system.

user number: Number from 0 to 15 assigned to a file when it is created. User numbers can organize files into sixteen file groups.

utility: Tool. Program that enables the user to perform certain operations such as copying files erasing files and editing files. Utilities are created for the convenience of programmers and users.

wildcard characters: Special characters that give CP/M Plus a pattern to match when it searches the directory for a file. CP/M Plus recognizes two wildcard characters ? and *. The ? can be substituted for any single character in a filespec and the * can be substituted for the primary filename or the filetype or both. By placing wildcard characters in a filespec you create an ambiguous filespec and can quickly reference one or more files.

Index