CP/M & CP/NET PROGRAMMER'S GUIDE

PN 12084

PREFACE

This publication is intended to help you write programs which run under the control of either CP/M or CP/NET for the Research Machines 380Z, 480Z and Network stations. Before reading it you should have read the relevant manuals described below and you should be familiar with the equipment you use.

480Z:	480Z Disc System Users Guide, PN 11900
380Z:	380Z-D Disc System Users Guide, PN 12163
Network stations:	Network Release 2.1 Users Guide, PN 12262

You should also have read the following manual:

CP/M Operating System Version 2.2D Users Guide, PN 11901

The BDOS functions described in this manual are mostly applicable to Z80 assembly language programmers. However, some high level languages such as BCPL will also allow you to exploit them and you should check the appropriate language manual for details of this facility.

Wherever CP/M is used in this manual, it refers to the CP/M Operating System. This, and CP/NET, are trademarks of Digital Research.

Parts of Chapters 2 and 4 of the current manual have been taken from Digital Research publications with their permission. We thank them for their cooperation.

Where the term Z80 is used in this manual, it refers to the Z80 microprocessor. This is a registered trademark of Zilog, Inc. Where the terms FORTRAN (80) and M80 Assembler are used, they refer to products of Microsoft.

CONTENTS

CHAPTER 1	INTRODUCTION	1.1
	The structure of CP/M	1.1
	The BIOS and BDOS	1.2
	The CCP	1.2
	The TPA	1.2
	CP/M interfaces	1.2
	Programming interfaces	1.2
	The user interface to the CCP	1.3
	Program development	1.4
	Deciding where to put your programs in memory	1.4
	Producing program source	1.5
	Assembling your programs	1.5
	Loading and running your programs	1.6
	Debugging your programs	1.7
CHAPTER 2	CP/M 2 BDOS FUNCTIONS	2.1
	Introduction	2.1
	Using simple BDOS functions	2.3
	Input/output to simple devices	2.3
	Miscellaneous routines	2.7
	Using BDOS disc functions	2.7
	Overview	2.7
	File maintenance	2.8
	Writing and reading files sequentially	2.9
	Random file operations	2.12
	Passing filenames to your programs using the keyboard	2.13
	Directory operations	2.14
	Disc and file protection	2.14
	Miscellaneous disc operations	2.15
	Transporting software from one machine to another	2.16
	Operating system functions	2.17
CHAPTER 3	DEBUGGING YOUR PROGRAMS	3.1
	Patching programs using the Front Panel	3.1
	Patching using the CP/M DDT utility	3.2
	Initiating DDT	3.2
	Operating DDT	3.3

CHAPTER 4 CP/NET BDOS FUNCTIONS 4.1

Introduction 4.1
How a station works 4.3
Modifying programs to run under CP/NET 4.4
Using the special facilities of CP/NET 4.5
Error handling under CP/NET 4.5
Device mapping across the network 4.9
Password protection under CP/NET 4.11
Temporary filename translation 4.12
CP/NET BDOS functions 4.13

CHAPTER 5 MORE ADVANCED USE OF CP/M 5.1

Adding a device handler 5.1
Running CP/M in a different area of memory 5.1
Connecting your device handler to CP/M 5.1
BIOS entry point subroutines 5.4
Disc parameter tables 5.6
Reserved locations in page zero 5.10

APPENDIX A A SAMPLE FILE-TO-FILE COPY PROGRAM A.1

APPENDIX B A SAMPLE FILE DUMP UTILITY B.1

APPENDIX C A SAMPLE RANDOM ACCESS PROGRAM C.1

APPENDIX D USE OF MEMORY BY CP/M D.1

INDEX I.1

CHAPTER 1

INTRODUCTION

The first two sections of this chapter describe the structure of CP/M and its interfaces. The last section describes the various phases of program development. Where necessary, you are pointed to further reading on particular topics.

An introduction the the Research Machines network and CP/NET is given in Chapter 4.

THE STRUCTURE OF CP/M

This section describes CP/M (version 2) system organization, including the structure of memory and system entry points. It provides the information you need to write programs which operate under CP/M and which use the peripheral and disc input/output (I/O) facilities of the system. The interface between CP/M and the Research Machines monitors (COS and ROS) is described in the following publication:

380Z/480Z Firmware Reference Manual, PN 10971

CP/M is divided into four logical parts, or modules:

1 The Basic Input/Output System (BIOS)

2 The Basic Disc Operating System (BDOS)

3 The Console Command Processor (CCP)

4 The Transient Program Area (TPA)

Their position in memory is shown in Figure 1.1.

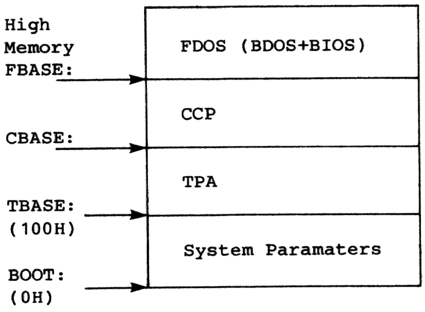


Figure 1.1 Organization of CP/M

The BIOS and BDOS

The BIOS defines the exact low level interface with a particular computer system which is necessary for peripheral device I/O; it is specific to the hardware of the computer concerned. The BDOS provides file structure and access to system devices, for example the keyboard and screen. They are both logically combined into a single module, with a common entry point, and this is referred to as the FDOS (see Figure 1.1).

The CCP

The CCP acts as an interface between the computer and the person operating its keyboard. It analyzes commands that you type on the keyboard (for example, ERA) and calls upon the FDOS to carry out the operations requested.

The TPA

The TPA is an area of memory where you can load and run your programs; it is also used to hold, during their period of operation, various non-resident operating system utilities (for example, PIP).

CP/M INTERFACES

Programming interfaces

Figure 1.2 shows you CP/M's interfaces.

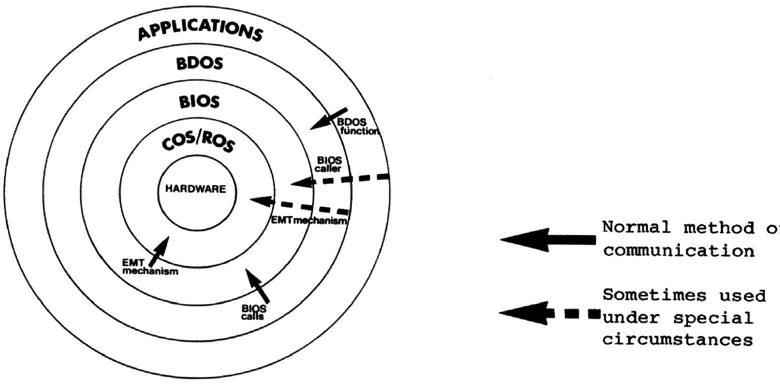


Figure 1.2 CP/M interfaces

At the highest level is the interface between your programs (application programs) and the BDOS. Your programs can access this using BDOS functions and this is the recommended method of programming under CP/M. Chapter 2 of this manual tells you all about these functions and how to use them. In addition, Chapter 4 tells you about the extra BDOS functions that are

At the next level is the interface between your programs and the BIOS. Again, you should not use this interface unless there is an alternative, but for different reasons. Basically, your BDOS calls go through the BIOS; if your programs access the BIOS directly they will go through the same mechanism as the BDOS calls and might corrupt the data structures in the BIOS.

At the lowest level, your programs can access the facilities of the Firmware, using EMT instructions. These are similar to the "software interrupts" or "emulator traps" of other computers and you should only use them if you have no alternative; they require a lot of understanding, and programs using EMT instructions will only run on Research Machines computers. If you need to use EMTs they are described in the following Research Machines publication:

380Z/480Z Firmware Reference Manual, PN 10971

The user interface to the CCP

You communicate with CP/M, via the CCP, by typing command lines such as DIR and REN after each prompt. Each command line takes one of the forms:

```
command

command file1

command file1 file2
```

where "command" is either a built-in function, such as DIR or TYPE, or the name of a transient command or program. Suppose the form:

```
command
```

is used. If the command is a built-in function of CP/M, it is executed immediately. Otherwise, the CCP searches the currently addressed disc for a file with the name:

```
command.COM
```

which contains a program. If it finds one, the program will be loaded.

If you try to load a transient program from a .COM file and the file is found, it is assumed to contain the memory image of a program that executes in the TPA and thus starts at 100H (TBASE) in memory. The CCP loads the COM file from the disc into memory starting at 100H and it can extend up to CBASE.

If the command is followed by one or two file specifications, the CCP prepares one or two file control block (FCB) names in the system parameter area.

The CCP passes control to the transient program and this begins execution. The transient program is "called" from the CCP; thus, provided it preserves the CCP stack, it can simply return to the CCP upon completion of its processing, or it can jump to BOOT to pass control back to CP/M. In the first case, the transient program must not use memory above CBASE, while in the latter case, memory up to FBASE-1 can be used.

PROGRAM DEVELOPMENT

The various phases of program development are summarized below:

1 Identifying and analyzing the problem

2 Designing your program

3 Deciding where to put your program in the computer's memory

4 Producing program source

5 Assembling your program (where necessary)

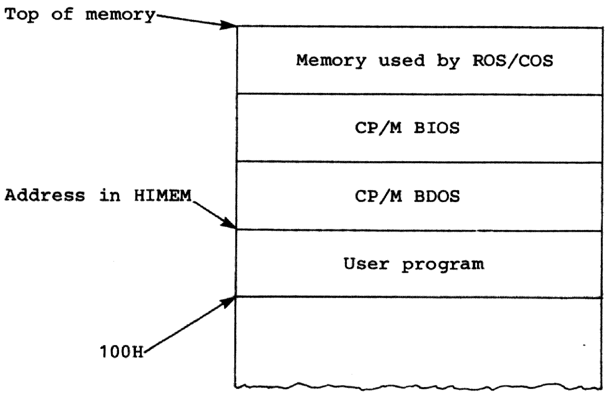
6 Loading and running your program

7 Debugging your program if it fails

The first two points will not be discussed here; they are worthy of a book in themselves. Instead, the rest of this section describes points 3 to 7 in more detail, pointing you to further reading on particular topics, where necessary.

Deciding where to put your programs in memory

Your programs must start at 100H and you can find out how much space you have available using the memory pointer HIMEM (at locations 006H and 007H). HIMEM contains the address shown in the diagram below; it is important to note that this address is the first unavailable address of memory.



The exact memory addresses of areas shown in Figure 1.1 vary from version to version. All standard CP/M versions, however, assume that BOOT = 0000H, the base of random access memory. The machine code found at location BOOT performs a system "warm start"; this loads and initializes the variables necessary to pass control to the CCP. Thus, transient programs need only jump to location BOOT to return control to CP/M at the command level. In the Research Machines version of CP/M, TBASE = 100H.

The principal entry point to the FDOS is at location 0005H, where a jump to FBASE is found.

A summary of the memory used by CP/M is given in Appendix D. The memory used by the ROS and COS Firmware is shown in detail in the following publication:

380Z/480Z Firmware Reference Manual, PN 10971

It has been necessary to reserve some memory for the COS or ROS Firmware and its work area. As a result of this, the largest CP/M systems you can run on a 380Z are as follows:

Type of installation	Size of largest CP/M system (includes CP/M)
32K 380Z system without HRG	31K
32K 380Z system with HRG	47K when HRG is not in use (includes 16K from graphics board)
64K 380Z or 480Z system	56K

Producing program source

When you use BASIC you produce your program source without the need for additional text editors. With other languages such as ZASM, the text editor TXED is available. This offers a wide range of functions and is described in detail in the Research Machines publication

TXED Text Editor and Formatter for Disc Systems, PN 11042

Assembling your programs

The ZASM Assembler was written by Research Machines and its use is recommended. It is described in the Research Machines publication:

ZASM Assembler for Disc and Network Systems, PN 11066

ZASM will assemble a source program to produce a file in industry standard

ZASM will assemble a source program to produce a file in industry standard Intel "hex" format, x.HEX. On Version 4.1J or later it can directly produce .COM files and .REL files.

You can also use the Microsoft M80 Assembler which is provided as part of the FORTRAN (80) package.

Loading and running your programs

The LOAD command reads a file (which is assumed to contain "HEX" format machine code) and produces a memory image file that can subsequently be executed. The file name is assumed to be of the form:

```
        X.HEX
```

and only the primary filename X need be specified in the command. The memory image file created is named:

```
        X.COM
```

The file is actually loaded into memory and executed when you type the filename X immediately after the prompt character ">" printed by the CCP.

Generally, the CCP reads the filename X following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the current default disc directory for a file with the name:

```
        X.COM
```

If found, the machine code is loaded into the TPA, and the program executes. Thus, you need only LOAD a hex file once; it can be subsequently executed any number of times by typing the primary name. In this way you can "invent" new commands in CP/M.

If you were to type:

```
        LOAD B:BETA
```

the situation is slightly different. Here, the "HEX" file will be created on the drive B, as requested, but if you want to load it into memory you must type:

```
        B:BETA
```

You should note that the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ZASM program, for example) and these must begin at 100H of the TPA. The addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hexadecimal records are read. Thus, LOAD must be used only for creating CP/M standard "COM" files that operate in the TPA. Programs which occupy regions of memory other than the TPA are loaded under DDT.

Most high-level languages (for example, FORTRAN) have compilers which produce a relocatable (.REL) form of machine code; ZASM can also produce this type of code. Several of these .REL files can be linked together using a linkage editor to produce a .COM file.

Debugging your programs

The 480Z and 380Z both contain a powerful debugging tool - the Front Panel. This contains a wide range of features to help you debug a machine code program, plus some machine control commands. A detailed description of how to use the Front Panel is given in the following manuals:

380Z/480Z Machine Language Programming Guide, PN 11068
380Z/480Z Firmware Reference Manual, PN 10971

Chapter 3 of the current manual describes how to patch your programs using the Front Panel and also mentions some of the other debugging facilities which are available.

CHAPTER 2

CP/M 2 BDOS FUNCTIONS

There are a number of standard operations which any program may want to perform, for example, getting characters from the keyboard, outputting to the screen and handling files. Operations like these are handled by the operating system and you can access them using the BDOS function calls. The early parts of this chapter describe how you can use these function calls in your programs and the last part gives a detailed description of each one.

INTRODUCTION

The complete range of BDOS function calls is shown in Table 2.1. Whenever you want to use one of them you do so in a well-defined way: your program must load the Z80 set of registers with parameter values; it must then make a machine code subroutine call to a specific address (0005H) which contains a jump to the operating system. This is the **only** defined entry point to the operating system.

In some cases, when the operating system has performed the operation you want, it uses the Z80 registers to pass information back to your program: for example, data from the keyboard or a warning that the operation was not successful.

Function No.	Function	Function No.	Function
0	System reset	20	Read sequential
1	Console input	21	Write sequential
2	Console output	22	Make file
3	Reader input	23	Rename file
4	Punch output	24	Return login vector
5	List output	25	Return current disc
6	Direct console I/O	26	Set DMA address
7	Not supported	27	Get address (alloc)
8	Not supported	28	Write protect disc
9	Print string	29	Get R/O vector
10	Read console buffer	30	Set file attributes
11	Get console status	31	Get address (disc parameters)
12	Return version number	32	Set/get user code
13	Reset disc system	33	Read random
14	Select disc	34	Write random
15	Open file	35	Compute file size
16	Close file	36	Set random record
17	Search for first entry	37	Reset drive
18	Search for next entry	38/39	Reserved
19	Delete file	40	Write random with zero fill

Table 2.1 BDOS functions

Before you do the call to the operating system you must load the Z80 registers as follows:

1 A function code must be loaded into register C; this tells the operating system which operation you want. If you want to output a character to the screen (console), for example, you would use function code 2 (the relevant function codes are shown in Table 2.1)

2 Any parameters which you want to transfer to the operating system must be loaded into some of the other Z80 registers (typically D and E). For example, if you wanted to print a character on the screen you need to tell the operating system which character to print

Since CP/M makes extensive use of the Z80 registers and does not save them beforehand, you should **save** the registers which you are using in your program by pushing them onto the stack and popping them on return from the function call. The program counter and stack pointer are left in a clean state by CP/M on exit.

Before we leave the subject of registers, two other points should be noted.

First of all, when you pass control to the BDOS function it will use **whatever information** is in the registers. If you have not set these up correctly it will still use the information they contain and the results may be rather embarrassing for you!

Secondly, the error handling information which CP/M passes back to you is limited and this makes it difficult to write good, fault-tolerant software. You should, therefore, make good use of any information passed back from CP/M: In general, the value -1 in the A-register means that an operation has failed, anything else usually means that it was successful. This is, however, not always the case, and you should refer to the description of the relevant function at the back of this chapter to be sure. If you want to write fault-tolerant software to run under CP/M you must write it using EMT instructions; these are described in the following Research Machines publication:

380Z/480Z Firmware Reference Manual, PN 10971

The BDOS function calls are designed for use with assembler language programs. However, some high level languages allow you to have a procedure which handles the passing of parameters to the BDOS. Even if the high level language does not have this facility, it may allow you to insert assembler language modules into your program and these could call BDOS functions. To find out about these facilities, you should refer to the relevant language manual (note, however, that you also need to know about BDOS functions).

Basically, the BDOS function calls can be separated into two groups. The first group handles input/output to the console, printer and other simple devices; it also performs a number of system maintenance functions such as resetting the system and identifying the system version number. These functions are described below under the heading "Using simple BDOS functions".

The second group of BDOS functions allows you to access discs: to write data to a disc, read from it, update it and look at the contents of the directory, for example. This group is described below under the heading "Using BDOS disc functions".

USING SIMPLE BDOS FUNCTIONS

Input/output to simple devices

Most programs need to accept data from the keyboard and write to the screen. If you are writing in a high level language, the language will contain routines which call the BDOS to do this for you. If you are writing in assembler language you may want to use BDOS functions.

One of the first things you would need to do is output a message to the console; you could do this using Function 2 (Console output). The following piece of code shows how you could output one character, the letter "A", to the console:

```
BDOS                =      0005H
FN_CONSOLE_OUT      =      2
                           .
                           .
                           .
                    LD     E,'A'
                    LD     C,FN_CONSOLE_OUT
                    CALL   BDOS
                           .
                           .
```

The character you want to transfer must be loaded into the E register and the function code must be loaded into the C register. Notice how the value "FN_CONSOLE_OUT" is used to load the C register and not the value "2". This makes the code more meaningful and easier to maintain.

The type of code shown above will be used in the examples in this chapter as it is easy to follow. However, if you are using ZASM version 4.1J or a later version, the following equivalent code is an example of better programming practice and is easier to support. It also highlights the fact that a call to the BDOS has been made.

```
     BDOSE             =      0005H
     FN_CONSOLE_OUT    =      2

     MACRO             BDOS,P1
     *L OFF
                       LD     C,P1
                       CALL   BDOSE
     *L ON
     ENDM
                              .
                              .
                              .
                       LD     E,'A'
                       BDOS   FN_CONSOLE_OUT
                              .
                              .
```

If you want to output a string of characters to the console, Function 9, (Print string) would be more useful than Function 2. This will print a string of text which is terminated by a "$" character. The following piece of code shows how it is used:

```
     BDOS              =      0005H
     FN_PRINT_STRING   =      9
     END_OF_STRING     =      '$'
     MESSAGE           DEFM   'This is a simple message'
                       DEFB   END_OF_STRING
                            .
                            .
                            .
                       LD     DE,MESSAGE
                       LD     C,FN_PRINT_STRING
                       CALL   BDOS
                            .
                            .
```

Here, the address of the message must be loaded into the DE registers before the call. The value "$" could have been put on the line following the string instead of "END_OF_STRING". Again, it is done this way for ease of maintenance; if the terminator character were to change, the program modifications would be minimized.

One problem with this function is its inability to print dollar characters ($). If you need to do this you should use a different function (Function 2).

As well as being able to send information to the console you will need to accept input from it; there are two aspects to this:

1 In some cases you will want to "echo" text input at the console to the screen, in some cases you will not. An example of the last case is the situation where a user types in a password; it is better not to echo this to the screen in case someone is looking over his or her shoulder!

2 In some cases you will want to suspend your program until text has been input, in other cases you will want to test if a character has been input but continue processing if it has not. You would want to do this in any arcade-type games program, for example

The simplest method of reading data from the console is by using Function 1, (Console input). This waits until a key is pressed on the console and then returns control to your program. On return from the call, the A register contains a single character and the character is echoed to the screen.

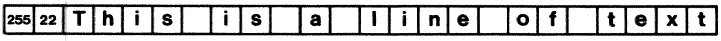
The following code shows how this function is called:

```
BOOT                =      0
BDOS                =      0005H
FN_CONSOLE_IN       =      1
                       .
                       .
                       .
NEXTC               LD     C,FN_CONSOLE_IN
                    CALL   BDOS                ;Return character in A
                    CP     '*'                 ;End of processing?
                    JR     NZ,NEXTC            ;Loop if not
                    JP     BOOT                ;Reload CP/M
                    END
```

Here, the program will read all characters input until the character "*" is typed. It would be useful if you just wanted to read one character from the keyboard but if you wanted to read a number of characters you would find Function 10 (Read console buffer) more useful. The following code shows this in use:

```
BDOS                       =      0005H
FN_READ_CONSOLE_BUFFER     =      10
BUFFER_LENGTH              =      255
BUFFER                     DEFB   BUFFER_LENGTH
                           DEFS   BUFFER_LENGTH+1
                                  .
                                  .
                                  .
                           LD     DE,BUFFER
                           LD     C,FN_READ_CONSOLE_BUFFER
                           CALL   BDOS
                                  .
                                  .
```

Any text input will be stored from BUFFER+2 onwards and the number of characters input will be stored in BUFFER+1 (this is shown in the diagram below).



You must specify the length of the buffer (1 to 255 characters) and this function reads all characters input until you press the RETURN key or until the buffer is full. If you type more characters than the length of the buffer allows, the excess characters will not be read and BDOS will return control to your program with an incomplete line. Also, if you do not reserve storage for the buffer, CP/M will overwrite your program; so be warned!

A particular advantage of the "Read console buffer" function is its simple editing facilities: these allow you to delete characters and backspace one character, for example. The full range of facilities is described in the last section of this chapter.

The console input functions described above are fine if you are prepared to suspend your program until the relevant character(s) are typed. However, in an arcade-type games program this is of no use. In such a situation, Function 11 (Get console status) will be of help. This checks to see if a character has been typed at the console. If it has, the value 0FFH is returned in register A; if it has not, a zero value is returned. Once you know that a character has been typed, you can use either Function 1 or Function 10 to get it.

Another disadvantage of the console input functions described above is the fact that they recognize <CTRL/C> and other operating system functions. Thus, if you inadvertently type <CTRL/C> you will re-boot the operating system. You can get round this problem by using Function 6 (Direct console I/O); this does not recognize operating system functions and it can be used for input or output. In addition, you can replace the two operations:

Get console status

Console input

with one call to Function 6.

A disadvantage of Function 6 is the fact that you cannot output graphics characters or "-1" with it.

The functions described above handle input/output to the console. Output to the printer is handled by Function 5 (List output) and this is very similar to Function 2 (Console output).

CP/M has two additional functions for simple devices:

Reader input

Punch output

These functions stem from the days when slow devices such as paper tape readers and punches were the only method of data input other than via the console. On the 480Z and 380Z, the reader is mapped to the SIO-4 port and the punch is mapped to a null device. You can interface your own devices to the reader and punch "slots", if you wish; the procedure is defined in Chapter 5.

Miscellaneous routines

There are a number of useful functions which do not relate to simple device handling or disc handling; they include Function 12 (Return version number) and Function 0 (System reset).

"Return version number" will tell you the version of CP/M under which your program is running; its use is recommended. If you do not use it and you are running under a version of CP/M earlier than 2.2, for instance, the use of some CP/M 2.2 facilities such as function 6, "Direct console I/O", will make your program crash. If you do use Function 12, you could either bypass the code which uses these facilities or stop the program running. Note, however, that CP/M 1.4 does not support "Return version number".

"Return version number" should always be used when writing programs to run under CP/NET; the BDOS functions described in Chapter 4 will only work under CP/NET.

Function 0, (System reset) reloads the BDOS and CCP part of CP/M but does not corrupt the TPA. It has the same effect as a jump to location BOOT and is the recommended way of exiting from your program.

USING BDOS DISC FUNCTIONS

Overview

Disc storage is one of the most significant features of CP/M and disc handling comprises the larger part of the BDOS functions.

Disc storage is similar, in some respects, to filing cabinet storage. The disc takes the place of a filing cabinet and inside it there are a number of "slots", or files, which can hold data. The filing cabinet has an "index" and the disc has the same, in the form of a "directory".

A file in CP/M can be thought of as a sequence of up to 65536 "records" of 128 bytes each, numbered from 0 through 65535; this allows a maximum of 8 megabytes per file. Note that if you want to write data in elements, or "sub-records", of less than 128 bytes, you must pad your data out to form complete records of 128 bytes yourself; CP/M will not do this for you, although some high level languages may.

Files are set up by a well-defined procedure. In practice, you would first "create" the file and then put data into it in the form of "records", written sequentially. The "create" or "make file" operation sets up an entry for the file in the disc's directory.

Finally, you would "close" the file; this operation updates the disc directory to describe the file. You must close the file, otherwise the results of your write operations will not be recorded in the directory and you will not be able to read them again. In addition, when using double density discs, disc transfers are buffered in the IDC and may not be written away until the file is closed.

Now that you have written a file you will, of course, want to read it back! The simplest way of doing this is to read it back in the order in which it was written, that is, sequentially.

To do this, you must first open the file to extract a description of its position from the directory. Next, you should read the file a record at a time. When you have finished manipulating the information in the file you should finally close it.

If you want to read or update records of your file in random order, the open and close operations are similar but in between you use read random and write random operations. Note that this is the only time you should use the write random operation; **files should not be created using this operation.** CP/M will cope with a file created in this way but the CP/M utilities may produce unexpected results.

You might think that you do not need to close a file which you are merely reading. In CP/M this is a valid assumption, but leaving a file open is not good programming practice; it is untidy. Apart from this, if your programs are ever likely to be run under CP/NET they are unlikely to work properly unless you close all files, even read-only files.

There are a number of other disc operations which you might want to perform, for example, renaming a file, deleting a file or searching the directory to find out if a file exists. These are described in more detail later.

File maintenance

Under CP/M a file can contain any number of records up to the full capacity of the drive; each drive is logically distinct, with a disc directory and file data area. The disc file names are in three parts: the drive select code; the filename and the filetype.

Source files 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, one 128-byte CP/M record could contain several lines of source text.

In a text file, <CTRL/Z> is used to pad the last record. The occurrence of <CTRL/Z> in text files denotes the end of file to most utilities; your program should deal with it as required.

Binary files (for example .COM files) contain an integral number of 128-byte records. <CTRL/Z> characters embedded within them are not recognized as end-of-file characters; the end-of-file condition returned by CP/M is used to indicate that the last record has been read.

In general, files in CP/M can be thought of as a sequence of records of 128 bytes each. However, you should note that although the records may be considered logically contiguous, they may not be physically contiguous in the disc data area. Internally, all files are divided into 16K byte segments called logical extents. The division into extents is discussed in subsequent paragraphs; however, they are not particularly significant for the programmer since each extent is automatically located by CP/M in both sequential and random access modes.

CP/M uses two memory blocks to pass data and information back and forth between your program and the disc. The first block is a 128 byte disc data buffer; you put any data to be written to the disc into this and, when you read from the disc, the data buffer will hold the result. A default location is provided for the data buffer at location 0080H. The start address of this buffer is known as the Direct Memory Access (DMA) address and you can alter it by using Function 26.

The second block is the File Control Block (FCB) and you use this to pass control information across to the BDOS. The format of the FCB is shown in Figure 2.1 and details are given in Table 2.2.

You must set up the FCB yourself and pass its address to the operating system in the DE register. The FCB consists of a sequence of 33 bytes for sequential access and 36 bytes for random access. It is a good idea to always reserve 36 bytes for each FCB, whatever its type; for random access operations, CP/M will use 36 bytes even if you have allocated 33! If you wish, you can use the default FCB at location 005CH.

Each file being accessed through CP/M must have a corresponding FCB which provides the name and allocation information for all subsequent file operations. When opening or creating files, it is your responsibility to fill the lower 13 bytes of the FCB and initialize some of the other fields. Normally, bytes 1 to 11 (fields 2 and 3) are set to the ASCII character values for the file name and file type, while all other fields are zero. **You must not alter the FCB contents after the open or create function; if you do so the system may crash.**

Once a file has been created, the information in its FCB is stored in a directory area of the disc. When you subsequently open the file, this information is brought into central memory before you proceed with file operations. The FCB is updated as file operations take place and the information is recorded permanently on disc at the termination of the file operation (see the CLOSE function).

Writing and reading files sequentially

Now that the FCB and disc data buffer have been described we will look at how they are used in disc operations. We will start with the simplest form of operation: writing a sequential file.

You access the BDOS disc function calls in a similar way to the simple BDOS functions: register C holds the function code and parameters are passed using the other registers. When you use Function 22 (Make file) to create your sequential file, you load the first 12 bytes of the FCB with the drive number, the filename and the file type, and store zeros in the rest of the FCB. You then pass the address of the FCB to the operating system using register pair DE and CP/M attempts to store the information in the FCB in the directory.

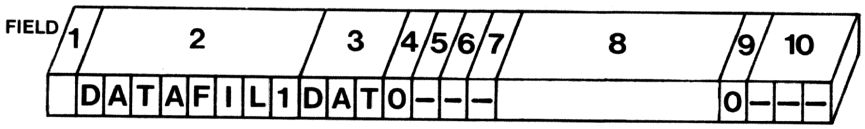


Figure 2.1 The structure of an FCB

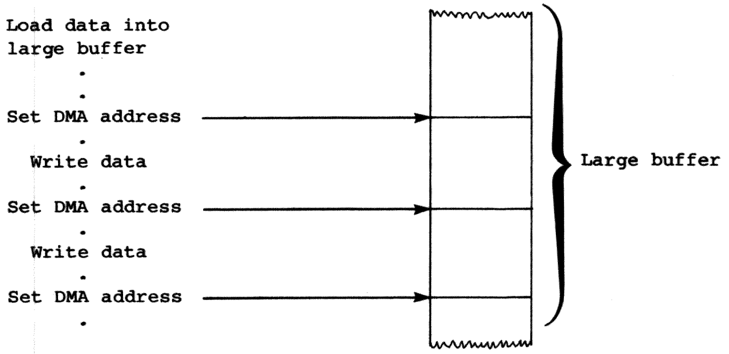
Field	Bytes	Description
1	0	Drive code (0-16) 0 means use default drive for file 1 means drive A, 2 means drive B, ... 16 means use drive P.
2	1-8	Contain the file name in ASCII upper case with high bit = 0
3	9-11	Contain the file type in ASCII upper case, with high bit = 0
4	12	Contains the current extent number, normally set to 00 by you, but in range 0-31 during file I/O
5	13	Reserved for internal system use
6	14	Reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH
7	15	Record count for the extent in field 4; takes on values from 0-127
8	16-31	Filled-in by CP/M, reserved for system use
9	32	Current record to read or write in a sequential file operation, initially set to zero by you
10	33-35	Optional random record number in the range 0-65535, with overflow to byte 35. Bytes 33 and 34 constitute a 16-bit value with low byte in byte 33 and high byte in byte 34

Table 2.2 The format of an FCB

If CP/M was able to create an entry in the directory for your file, it will return the value 0, 1, 2 or 3 in the A register; otherwise, it will return the value -1. The following piece of code shows how you could check this information:

```
     BDOS              =      0005H
     FN_MAKE_FILE      =      22
     FCB:
                              .
                              .
                              .
                       LD     C,FN_MAKE_FILE
                       LD     DE,FCB
                       CALL   BDOS
                       INC    A
                       JP     Z,FAIL
                              .
                              .
     FAIL:                    .
                              .
                              .
```

The next operation you would want to perform is to write blocks sequentially to your file. To do this, you must first decide upon the address of your disc data buffer. You could use the default data buffer at address 0080H; however, if you want to write several blocks to the disc, it would be better to organize them sequentially in memory; you can then alter the data buffer address to the beginning of each block of memory before each write operation. You do this using Function 26 (Set DMA address) and the entire operation is shown in the diagram below:



Once you have set the DMA address, you can do your write operation using Function 21 (Write sequential). Again, you pass the address of the (same) FCB to CP/M in register pair DE. You should be careful not to clear or otherwise modify the FCB; if you do this, CP/M will lose its file pointers.

The final operation you perform when writing your sequential file is to close the file, using Function 16 (Close file). Again, you pass the address of the FCB to CP/M using register pair DE. If the operation is unsuccessful, register A will contain the value -1; if it is successful, the register will contain a number from 0 to 3.

To read your file sequentially, you would first open the file using Function 15 (Open file); this is similar in concept to Function 22 (Make file). You would then set the DMA address using Function 26 and then read the file a block at a time using Function 20 (Read sequential). Again, it is most efficient to read the data in the manner shown above for the write operation: first set the DMA address to the start of a large block of memory, read a block and then alter the DMA address to the start of the next area of memory.

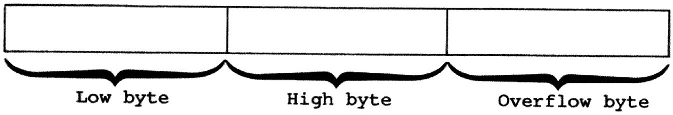
If you want to append extra blocks to your file you can do this easily. You would first open your file, then read up to the end of it and finally write the new blocks using Function 21 (Write sequential).

Appendix A shows a file-to-file copy program which uses the sequential file access functions.

Random file operations

The read random (Function 33) and write random (Function 34) file operations need 36 byte FCBs; they use the last three bytes of the FCB as a record pointer.

You call both of these functions in a similar way to the sequential read and write operations, but first you must set the required record number. This has the following format:



If the transfer was successful, a zero value will be returned in register A; if it was unsuccessful, an error code of value 1 to 6 will be returned. The explanation of these error codes is given under the description of the relevant function.

Once again, you are strongly recommended not to use Function 34 (Write random) when creating your files.

Passing filenames to your programs using the keyboard

When you load a program, you can pass the names of two files to it by typing a command such as:

```
PROGNAME  file1 file2
```

The file PROGNAME.COM is loaded into the TPA and the "command line tail" (denoted by "file1" and "file2" above) is stored in the default DMA buffer at location 80H. The first position contains the number of characters, with the characters themselves following the character count. If, for example, you type:

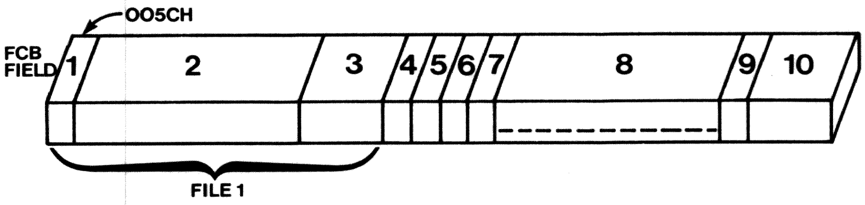
```
PROGNAME B:X.ZOT Y.ZAP
```

the area starting at location 80H will be initialized as follows:

+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+A	+B	+C	+D	+E
EH	' '	'B'	':'	'X'	'.'	'Z'	'O'	'T'	' '	'Y'	'.'	'Z'	'A'	'P'

The characters are translated to upper case ASCII with uninitialized memory following the last valid character. It is your responsibility to extract the information from this buffer before any file operations are performed. In particular, if you re-define the data buffer address using Function 26 (Set DMA address), you should note that command line tails will still be stored at location 80H onwards; they are not moved to your new address.

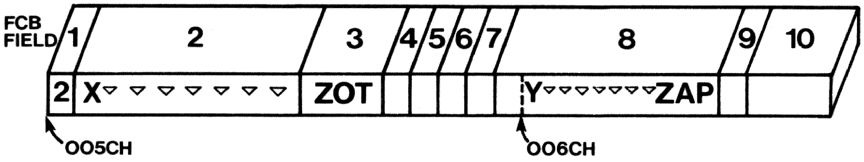
As well as being stored from location 80H onwards, the command line tail is also stored in the default FCB, though in a slightly different form. The first part of the information (file1) is stored at address 005CH, in fields 2 and 3 of the default FCB, the second part is stored within field 8, at address 006CH (see the diagram below). Notice that the dots and colons have been removed from the command line tail, in this case. Before your program can open these files, it must move the second part (file2) to another area of memory, otherwise it will be overwritten and lost.



In the following example:

```
PROGNAME B:X.ZOT Y.ZAP
```

the file PROGNAME.COM is loaded into the TPA, and the default FCB at 005CH is initialized to drive code 2, file name X, and file type ZOT (see the diagram below).



The second drive code takes the default value 0, which is placed at 006CH, with the file name Y placed into location 006DH and file type ZAP located 8 bytes later at 0075H. All of the remaining fields through field 9 are set to zero. You should note again that it is your responsibility to move this second file name and file type to another area (usually a separate file control block) before opening the file that begins at 005CH, because the open operation will overwrite the second filename and filetype.

If no file names are specified in the original command, the fields beginning at 005DH and 006DH contain blanks. In all cases, the CCP translates lower case alphabetics to upper case so as to be consistent with the CP/M file naming conventions.

Appendix B shows the listing of a file dump utility which allows you to pass file names to it from the console.

Directory operations

You will be familiar with the CP/M console commands which allow you to delete a file and rename a file. You can also perform these actions from your programs using Function 19 (Delete file) and Function 23 (Rename file). In fact, the console commands use these functions themselves.

In addition to the above commands, Function 17 (Search for first) and Function 18 (Search for next) allow you to search the directory for the existence of one or more files. You start with "Search for first" and put the filename which you want in an FCB. You do not need to give the exact filename: if you insert a "?" character at any point in the filename, CP/M will ignore this character position and return the first file whose name matches the rest of the characters in the FCB. This is known as a "wildcard facility". CP/M reads into your data buffer the directory block which contains the filename and also gives you a pointer to the directory entry.

If you are using the wildcard facility to search for a number of filenames, you can use Function 18 (Search for next) to get the next name which matches the FCB entry. Its action is similar to that of Function 17.

Disc and file protection

You can protect the information on your discs in various ways:

1 By using the physical write-protect facilities. The method depends upon the discs which you use; basically, it consists of either covering or uncovering the write-protect notch

2 By using a BDOS function to write-protect the entire disc

3 By using a BDOS function to write-protect a specific file

The physical write-protection facilities apply to all types of programs which access discs, even those using firmware routines. They are described fully in the Users Guide for your computer and you should refer to this for detailed information.

To protect the entire disc using software, you can use Function 28 (Write protect disc). This will temporarily protect the currently-selected disc until one of the following events occurs:

1 A "cold" or "warm" start operation

2 A disc reset

3 BDOS Fuction 13 (Reset disc system) is called

If you want to find out which discs are protected in this way you can use Function 29 (Get read/only vector). This returns a value in the HL register pair and each bit of this value corresponds to the status of a specific drive. Note that this method of protection is applicable only to programs using CP/M functions; it can be circumvented using firmware EMT calls.

File protection is indicated in the FCB by the status of the most-significant bit in the first byte of the filetype (see Table 2.2). This is called the "read-only file attribute" and if this bit is set, the file will be protected against writing; if it is clear, the file will not be protected. You set and clear the status bit using Function 30 (Set file attributes). You should not change these attribute bits other than with Function 30 (Set file attributes).

If you want to check to see if a file is protected, you should use Function 17 (Search for first). This will return the directory entry for the file in the form of an FCB; you can then look at the status bit and find out if the file is protected or not.

Miscellaneous disc operations

There are a number of disc operations which have not yet been covered and these are summarized below. In these descriptions the following conventions apply:

1 The currently-selected drive is the default drive for all file operations

2 Logged-in drives are all logical drives that have been accessed since the last cold or warm start, disc reset or call to Function 13 (Reset disc system)

Function 13 (Reset disc system) can be used to restore the drives which have been write-protected (using Function 28) to a read/write state. You can reset a specific drive using Function 37 (Reset drive).

Function 14 (Select disc) allows you to specify a disc as the default disc for subsequent file operations. Its converse, Function 25 (Return current disc) will tell you which is the current default drive.

Function 24 (Return current log-in vector) will tell you which drives are currently logged-in. If you want to find out what space is available on one of these discs you must first make the disc you want the current disc, then you can use Function 27 (Get ADDR (Alloc)). An allocation vector is maintained in memory for each on-line disc drive and this function can be used to interpret the information in the vector to give you the amount of remaining storage.

If you want to determine the size of a file you can do so using Function 35 (Compute file size). On return from the function, the random record bytes in the FCB contain the record address of the record following the end of the file. Thus, you could append data to the end of an existing file by merely calling Function 35 and then performing a sequence of random writes starting at the preset record address.

TRANSPORTING SOFTWARE FROM ONE MACHINE TO ANOTHER

There are two aspects to this subject: you might want to run software that was implemented on another computer upon your 380Z or 480Z; you might also want to write software on your 380Z or 480Z that is suitable for running on other machines, ie portable software. In fact, both are related: the points you look for when converting a program to run on the 380Z and 480Z are the same as the ones you should bear in mind when writing portable software.

The first thing to check in programs from other machines is whether or not they use memory in page zero; software that uses the C compiler often uses this area. In particular, the RST addresses must not be used; they are used by the firmware.

The next things which usually cause problems are screen attributes and control sequences; these are usually machine-dependent. If you are writing portable software it is best to put screen control sequences in a table and address each (variable-length) sequence using a separate table of pointers; this is then easy to change.

Other points that may give you problems when running programs from other machines are listed below:

1 Special function keys may have been used

2 The way in which the source machine handles colour will probably not match the way in which the 380Z or 480Z handles it

3 The disc space on your 380Z or 480Z may not be sufficient for the program

4 The program may make assumptions regarding the speed of the target machine in matters concerning timing (especially in the case of real-time programs)

5 The program may use a non-standard character set

6 There may be a need for special devices or special hardware

7 The memory of your 380Z or 480Z may not be large enough to hold the program

8 The program may be designed for a printer with a special character set

9 The use of machine-specific subroutines, for example, the use of EMTs

When writing portable software, many of these details can be set up in the form of a table and will thus be easier to change. You should also isolate BDOS calls, indeed any input/output calls, to individual subroutines. This gives an overhead but ensures that the program is more-suitable for transfer to non-CP/M systems.

There is always a trade-off between portability and performance when writing software. Sometimes it is more efficient to break the rules and take advantage of the special features of a machine. However, you should always remember that <u>you</u> might have to modify your program to make it run on a different machine.

OPERATING SYSTEM FUNCTIONS

Function 0: System Reset

Entry Parameters:
    Register C: 00H

The system reset function returns control to the CP/M operating system at the CCP level. The CCP reinitializes the disc subsystem by selecting and logging in disc drive A, then reselecting the default drive. This function has exactly the same effect as a jump to location BOOT.

Function 1: Console Input

Entry Parameters:
    Register C: 01H

Returned Value:
    Register A: ASCII Character

The console input function reads the next console character into register A. Graphic characters, along with carriage return, line feed, and back space <CTRL/H> are echoed to the console. Tab characters <CTRL/I> move the cursor to the next tab stop.

The FDOS does not return to the calling program until a character has been entered, thus suspending execution if a character is not ready. Function 1, therefore, is not suitable for real-time use. Instead, you should use Function 6 (Direct console I/O).

Function 2: Console Output

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

The ASCII character from register E is sent to the console device. As in Function 1, tabs are expanded and checks are made for <CTRL/S> (CP/M start/stop scroll).

Function 3: Reader Input

Entry Parameters:
  Register C: 03H

Returned Value:
  Register A: ASCII Character

The next character will be read from the logical read device into register A. Control does not return until the character has been read.

The "Reader Input" function is a throw-back to the days when input was mainly from paper tape. Currently, the reader device slot in CP/M is mapped to the SIO-4 port.

You can, if you wish, hook in your own devices to the reader slot in CP/M; to do this you will have to write your own device driver as described in the Research Machines publication:

380Z/480Z Firmware Reference Manual

Function 4: Punch Output

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

The "Punch Output" function sends the character from register E to the logical punch device.

This function is also a throw-back to the days of paper tape. If you try to output to the punch device your data will be discarded, unless you have written a device driver of your own to handle such transfers.

Function 5: List Output

```
Entry Parameters:
     Register C:  05H
     Register E:  ASCII Character
```

The "List Output" function sends the ASCII character in register E to the logical listing device.

Function 6: Direct Console I/O

```
Entry Parameters:
     Register C:  06H
     Register E:  0FFH (input) or
                  char (output)

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

Direct I/O is supported under CP/M for those specialized applications where basic console input and output are required. Use of this function should, in general, be avoided since it bypasses all of the normal CP/M control character functions (such as <CTRL/S> and <CTRL/P>. However, programs which perform direct I/O through the BIOS, under previous releases of CP/M, should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register E either contains hexadecimal FF, denoting a console input request, or an ASCII character. If the input value is FF, function 6 returns A = 00 if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, function 6 assumes that E contains a valid ASCII character which is sent to the console. The contents of the A register are undefined.

Function 6 must not be used in conjunction with other console I/O functions.

Function 9: Print String

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

The "Print String" function sends to the console device the character string which is stored in memory at the location given by DE. Characters are transferred until a $ is encountered in the string. Tabs are expanded as in Function 2, and checks are made for start/stop scroll.

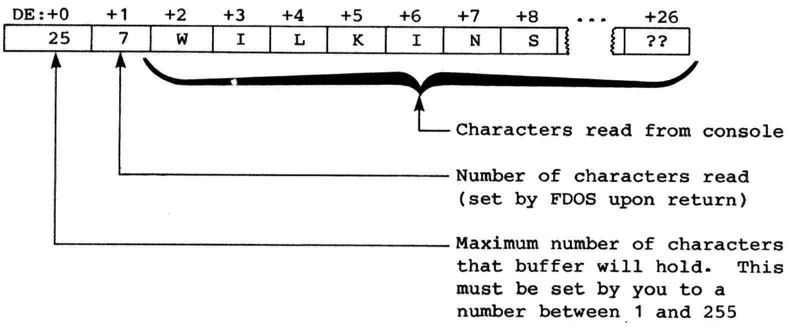
If you want to print text containing $ signs you should not use this function; instead, you should use Function 2 (Console out).

Function 10: Read Console Buffer

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

Returned Value:
Console Characters in Buffer

The "Read Buffer" function reads a line of edited console input into a buffer addressed by register pair DE. Console input is terminated either when the input buffer overflows or a carriage return or line feed is typed. The input buffer takes the form:



DE:+0
+1
+2
+3
+4
+5
+6
+7
+8
...
+26
25
7
W
I
L
K
I
N
S
??
Characters read from console
Number of characters read (set by FDOS upon return)
Maximum number of characters that buffer will hold. This must be set by you to a number between 1 and 255

If the number of characters that the buffer will hold is greater than the number of characters read, uninitialized positions will follow the last character, denoted by "??" in the above diagram. A number of control functions are recognized during line editing:

Control Function	Effect
<RUB/DEL>	Removes the last character and backspaces one character position
<CTRL/C>	Reboots when at the beginning of line
<CTRL/E>	Causes physical end of line
<CTRL/H>	Removes the last character and backspaces one character position
<CTRL/J> <LINE FEED>	Terminates input line
<CTRL/M> <RETURN>	Terminates input line
<CTRL/P>	Printer echo toggle
<CTRL/R>	Retypes the current line on the next line showing all the changes made
<CTRL/U>	Removes current line
<CTRL/X>	Same as CTRL/U.

You should also note that certain functions which return the carriage to the leftmost position (for example, <CTRL/X>) do so only to the column position where the prompt ended (in earlier releases, the carriage returned to the extreme left margin). This convention makes operator data input and line correction more legible.

Function 11: Get Console Status

Entry Parameters:
    Register   C: 0BH

Returned Value:
    Register   A: Console Status

The Console Status function checks to see if a character has been typed at the console. If a character has been typed, the value 0FFH is returned in Register A, otherwise, a 00H value is returned.

Function 12: Return Version Number

Entry Parameters:
    Register C: 0CH

Returned Value:
    Register pair HL: Version Number

Function 12 provides information which helps you write programs that are transferable between CP/M versions. Using function 12, for example, you can write application programs that provide both sequential and random access functions.

A two-byte value is returned, with H = 00 designating the CP/M release (H = 01 for MP/M) and L = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register L, with subsequent version 2 releases in the hexadecimal range 21, 22, up to 2F.

Function 13: Reset Disc System

Entry Parameters:
    Register C: 0DH

One of the other disc functions (Function 28) allows you to protect a disc against writing from within your programs. The "Reset disc system" function is used to restore the file system, by program, to a reset state where all discs are set to read/write. Only disc drive A is selected and the default DMA address is reset to 0080H.

This function could be used, for example, by an application program which requires a disc change without a system reboot.

Function 14: Select Disc

Entry Parameters:
    Register C: 0EH
    Register E: Selected Disc

The Select Disc function designates the disc drive named in register E as the default disc for subsequent file operations, with E = 0 for drive A, 1 for drive B, and so on through 15 (corresponding to drive P in a full 16 drive system). The drive is placed in an on-line status, and this activates its directory until the next cold start, warm start, or disc system reset operation. If the disc medium is changed whilst it is on-line, the drive automatically goes to a read-only status in a standard CP/M environment (see function 28). FCBs which specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference A through P.

Function 15: Open File

Entry Parameters:

Register	C:	0FH
Registers	DE:	FCB Address

Returned Value:

Register	A:	Directory Code

The "Open File" operation is used to open a file whose entry currently exists in the disc directory for the currently active user number.

The FDOS scans the referenced disc directory for a match in positions 1 through 14 of the FCB referenced by DE. If you put an ASCII question mark (3FH) in any character position in the filename, any directory character will be accepted. Normally, no question marks should be included.

If a directory element is matched, the relevant directory information will be copied into bytes 16 to 31 of the FCB, thus allowing access to the files through subsequent read and write operations. You should note that an existing file must not be accessed until a successful open operation is completed.

Upon return, the open function returns a directory code (see below) with the value 0 through 3 if the open was successful or 0FFH (255 decimal) if the file cannot be found. If question marks occur in the FCB, the first matching FCB is activated.

Note that the current record pointer (byte 32) must be zeroed by your program if the file is to be accessed sequentially from the first record.

The directory code is a value in the range 0 to 3 and it is returned in the A register; it can be used to find the directory entry, in the current DMA buffer, of your file. If you multiply the contents of the A register by 32 (ie shift the A register left by 5 bits) this will give you the start address of the directory entry in the buffer.

Function 16: Close File

Entry Parameters:

Register	C:	10H
Registers	DE:	FCB Address

Returned Value:

Register	A:	Directory Code

The Close File function performs the inverse of the open file function. Given that the FCB addressed by DE has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disc directory.

The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, whilst FFH (255 decimal) is returned if the file name cannot be found in the directory.

A file must be closed, even if only read operations have taken place; this ensures software compatibility with CP/NET networks and Research Machines network systems, in particular.

Function 17: Search for First

Entry Parameters:
    Register C: 11H
    Register pair DE: FCB Address

Returned Value:
    Register A: Directory Code

"Search First" scans the directory for a match with the file given in the FCB addressed by DE. This function has a wildcard facility: if you insert a "?" character in any positions within the filename, filetype or extent field, these positions will be ignored during the directory search on the default or auto-selected disc drive. If the drive code field contains a "?" character, the auto disc select function will be disabled and the default disc will be searched. In this case, the search function will return any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but it allows complete flexibility to scan all current directory values.

The value 255 (hexadecimal FF) is returned if the file is not found; otherwise, a directory code in the range 0 to 3 is returned and indicates that the file is present.

Function 18: Search for Next

Entry Parameters:
    Register C: 12H

Returned Value:
    Register A: Directory Code

The "Search Next" function is similar to the "Search First" function, except that the directory scan continues from the last matched entry. Function 18 also returns the decimal value 255 in A when no more directory items match.

You should not attempt any disc operations between calls to Functions 17 and 18.

Function 19: Delete File

Entry Parameters:
    Register C: 13H
    Register pair DE: FCB Address

Returned Value:
    Register A: Directory Code

The "Delete File" function removes files that match the FCB addressed by DE. The filename and type may contain wildcards (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as it may be in the "Search First" and "Search Next" functions.

Function 19 returns a decimal 255 if the referenced file or files cannot be found; otherwise, a value in the range 0 to 3 is returned.

Function 20: Read Sequential

Entry Parameters:
    Register C: 14H
    Register pair DE: FCB Address

Returned Value:
    Register A: Directory Code

Given that the FCB addressed by DE has been activated through an open or make operation (Functions 15 and 22), the "Read Sequential" function reads the next 128-byte block from the file into memory at the current DMA address.

The record is read from a position in the extent given by byte 32 of the FCB; this byte is then automatically incremented to the next record position. If byte 32 overflows, the next logical extent is automatically opened and byte 32 is reset to zero in preparation for the next read operation.

The value 00H is returned in the A register if the read operation was successful; a nonzero value is returned if no data exists at the next record position (for example, if end-of-file occurs).

Function 21: Write Sequential

Entry Parameters:
    Register C: 15H
    Register pair DE: FCB Address

Returned Value:
    Register A: Directory Code

Given that the FCB addressed by DE has been activated through an open or make operation (Functions 15 and 22), the "Write Sequential" function writes the 128-byte data block to the file from the memory at the current DMA address.

The block is placed at the position pointed to by byte 32 of the FCB, and byte 32 is automatically incremented to point to the next block position. If byte 32 overflows, the next logical extent is automatically opened and byte 32 is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case newly-written records overlay those which already exist in the file.

Upon return from a successful write operation, Register A = 00H; a nonzero value indicates an unsuccessful write caused by a full disc or hardware errors.

Function 22: Make File

Entry Parameters:
    Register C: 16H
    Register pair DE: FCB Address

Returned Value:
    Register A: Directory Code

The "Make File" operation is similar to the "Open File" operation, except that the FCB must name a file which does not exist in the currently referenced disc directory (i.e., the one named explicitly by a nonzero drive field, or the default disc, if the drive code is zero). The FDOS creates the file and initializes both the directory and main memory value to an empty file. The make function has the side effect of activating the FCB; thus, a subsequent open is not necessary.

You must ensure that no duplicate file names occur; a preceding delete operation is sufficient if there is any possibility of duplication.

Upon return, register A = 0, 1, 2, or 3, if the operation was successful, and 0FFH (255 decimal) if no more directory space is available.

Function 23: Rename File

Entry Parameters:
    Register C: 17H
    Register pair DE: FCB Address

Returned Value:
    Register A: Directory Code

The Rename function uses the FCB addressed by DE to change all occurrences of the file named in the first 16 bytes, to the file named in the second 16 bytes. The drive code at Byte 0 is used to select the drive, whilst the drive code for the new filename (at position 16 of the FCB) is assumed to be zero. Upon return, register A is set to a value between 0 and 3, if the rename was successful, and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

This function will not accept the wildcard character "?" in the FCB. You must also ensure that no duplicate filenames occur.

Function 24: Return Log-in Vector

Entry Parameters:
    Register C: 18H

Returned Value:
    Register pair HL: Log-in Vector

The log-in vector value returned by CP/M is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A and the high order bit of H corresponds to the sixteenth drive, labeled P. A 0 bit indicates that the drive is not on-line; a 1 bit marks a drive which is actively on-line as a result of an explicit disc drive selection, or an implicit drive selection caused by a file operation that specified a nonzero drive field. You should note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

Function 25: Return Current Disc

Entry Parameters:
    Register C: 19H

Returned Value:
    Register A: Current Disc

Function 25 returns the currently selected default disc number in register A. The disc numbers range from 0 to15, corresponding to drives A to P.

Function 26: Set DMA Address

Entry Parameters:
    Register C: 1AH
    Register pair DE: DMA Address

DMA is an acronym for Direct Memory Access; this is often used in connection with disc controllers which directly access the memory of the computer to transfer data to and from the disc subsystem. Although many computer systems use non-DMA access (i.e., the data is transferred through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128-byte data block resides in memory before a disc write and after a disc read operation.

Upon cold start, warm start, or disc system reset, the DMA address is automatically set to 0080H. The "Set DMA Address" function, can, however, be used to change this default value to address another area of memory where the data blocks reside. Thus, the DMA address becomes the value specified by DE until it is changed by a subsequent "Set DMA Address" function, cold start, warm start, or disc system reset.

Function 27: Get ADDR(Alloc)

Entry Parameters:
    Register C: 1BH

Returned Value:
    Register pair HL: ALLOC Address

An allocation vector is maintained in main memory for each on-line disc drive. Various system programs use the information which it provides to determine the amount of remaining storage (see the STAT program). Function 27 returns the base address of this allocation vector for the currently selected disc drive. However, the allocation information may be invalid if the disc selected has been marked as read-only.

Function 28: Write Protect Disc

Entry Parameters:
    Register C: 1CH

The "Write Protect Disc" function provides temporary write protection for the currently selected disc. Any attempt to write to the disc before the next cold or warm start operation will produce the message:

```
BDOS ERR on d: R/O
```

To clear the write protection, you must do one of the following:

1 A cold or warm start

2 Call Function 13 (Reset disc system)

3 Call Function 37 (Reset drive)

Function 29: Get Read/Only Vector

Entry Parameters:
    Register C: 1DH

Returned Value:
    Register pair HL: R/O Vector Value

Function 29 returns a bit vector in register pair HL; this indicates drives which have the temporary read-only bit set. As in function 24, the least significant bit corresponds to drive A, whilst the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to Function 28 or by the automatic software mechanisms within CP/M that detect changed discs.

Function 30: Set File Attributes

```
Entry Parameters:
     Register   C:  1EH
     Register pair DE:  FCB Address

Returned Value:
     Register   A:  Directory Code
```

The most significant bits in Bytes 1 to 11 of the FCB are referred to as the "file attributes"; currently, only those of Bytes 9 and 10 are used. The attributes of Bytes 1 to 4 can be used by application programs, since they are not involved in the matching process during file open and close operations. Those of Bytes 5 to 8 and Byte 11 are reserved for future system expansion.

The "Set File Attributes" function allows manipulation by program of these attributes. In particular, the R/O and System attributes (the most significant bits in Bytes 9 and 10 of the FCB) can be set or reset.

When using this function, the DE pair should address an unambiguous file name with the appropriate attributes set or reset. Function 30 searches for a match and changes the matched directory entry to contain the selected indicators.

Function 31: Get ADDR(Disc Parms)

```
Entry Parameters:
     Register   C:  1FH

Returned Value:
     Register pair HL:  DPB Address
```

The address of the BIOS-resident disc parameter block is returned in HL as a result of this function call. This address can be used in either of two ways. First, the disc parameter values can be extracted for display and space computation purposes. Secondly, transient programs can dynamically change the values of current disc parameters when the disc environment changes, if required. Normally, application programs will not require this facility.

Function 32: Set/Get User Code

```
Entry Parameters:
     Register C:  20H
     Register E:  0FFH (get) or
                  User Code (set)

Returned Value:
     Register A:  Current Code or
                  (no value)
```

An application program can change or interrogate the currently active user number by calling Function 32. If register E = 0FFH, the value of the current user number is returned in register A and is in the range 0 to 15. If register E is not 0FFH, the current user number is changed to the value of E (modulo 16).

Function 33: Read Random

Entry Parameters:
Register C: 21H
Register pair DE: FCB Address

Returned Value:
Register A: Return Code

The "Read Random" function is similar to the sequential file read operation of previous releases. However, the read operation takes place at a particular record number, selected by a 24-bit value specified in the FCB. This value is a 3-byte field starting at Byte 33 of the FCB. You should note that the sequence of 24 bits is stored with least significant byte first (byte 33), middle byte next (byte 34) and high byte last (byte 35). CP/M does not reference byte 35, except in computing the size of a file (function 35). Byte 35 must be zero, however, since a nonzero value indicates overflow past the end of file.

Thus, Bytes 33 and 34 are treated as a double-byte, or "word" value which contains the record to be read. This value ranges from 0 to 65535, thus providing access to any particular record of the 8-megabyte file. To process a file using random access, the base extent (extent 0) must first be opened using Function 15. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory and is visible in DIR requests. The selected record number is then stored in the random record field (Bytes 33 and 34) and the BDOS is called to read the record.

Upon return from the call, register A either contains an error code, as listed below, or the value 00, indicating that the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. You should note that, contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations will continue to read the same record.

During each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. However, you should note that, in this case, the last randomly read record will be reread as one switches from random mode to sequential read mode and the last record will be rewritten as one switches to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register A following a random read are listed below.

Error Code	Meaning
01	Reading unwritten data
02	(not returned in read mode)
03	Cannot close current extent
04	Seek to unwritten extent
05	(not returned in read mode)
06	File size overflow (byte 35 of FCB too big)

Error codes 01 and 04 occur when a random read operation accesses a data block which has not been previously written or an extent which has not been created; these are equivalent conditions. Error code 03 does not normally occur under proper system operation; if it does, it can be cleared by simply rereading extent zero as long as the disc is not physically write protected. Error code 06 occurs whenever Byte 35 of the FCB is nonzero. Normally, nonzero codes can be treated as missing data, with zero return codes indicating completion of the operation.

Function 34: Write Random

Entry Parameters:
    Register C: 22H
    Register pair DE: FCB Address

Returned Value:
    Register A: Return Code

The Write Random operation is initiated in a similar way to the Read Random call, except that data is written to the disc from the current DMA address. Further, if the disc extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues.

As in the Read Random operation, the random record number is not changed as a result of the write operation. The logical extent number and current record positions of the file control block are set to correspond to the random record that is being written. **Again, sequential read or write operations can begin following a random write, with the convention that the currently addressed record is either re-read or rewritten again as the sequential operation begins.** You can also simply advance the random record position following each write to get the effect of a sequential write operation.

Note, in particular, that reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

The error codes returned by a random write are identical to those for the random read operation, with two additions: error code 05 indicates that a new extent cannot be created as a result of directory overflow; error code 02 indicates that the disc is full.

Function 35: Compute File Size

Entry Parameters:
    Register C: 23H
    Register pair DE: FCB Address

Returned Value:
    Random Record Field Set

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes 33, 34 and 35 are present). The FCB contains an unambiguous file name which is used in the directory scan.

Upon return, the random record bytes contain the "virtual" file size, which is, in effect, the record address of the record following the end of the file. Following a call to function 35, if the high record byte (Byte 35) is 01, the file contains the maximum record count 65536. Otherwise, bytes 33 and 34 constitute a 16-bit value which is the file size (Byte 33 is the least significant byte, as before).

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If the file was created in random mode and "holes" exist in the allocation, the file may in fact contain fewer records than the size indicates. For example, if only the last record of an 8-megabyte file is written in random mode (i.e., record number 65535), the virtual size is 65536 records, although only one block of data is actually allocated.

Function 36: Set Random Record

Entry Parameters:
    Register C: 24H
    Register pair DE: FCB Address

Returned Value:
    Random Record Field Set

The "Set Random Record" function can be used to find out the position of a specific block when reading or writing a file sequentially. It can be useful in two ways:

First, it is often necessary to read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, Function 36 can be 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 position is placed into a table, together with the key, for later retrieval. After scanning the entire file and tabulating the keys and their record numbers, you can then move instantly to a particular keyed record by performing a random read, using the corresponding random record number which was saved earlier. You can easily generalize the scheme for variable record lengths, since the program need only store the byte position relative to the start of the buffer, 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 is when switching from a sequential read or write to random read or write. Here, the file is accessed sequentially to a particular point, Function 36 is called to set the record number, and subsequent random read and write operations continue from the selected point in the file.

Function 37: Reset Drive

```
Entry Parameters:
     Register    C:  25H
     Register pair DE:  Drive Vector

Returned Value:
     Register    A:  00H
```

The Reset Drive function allows you to reset specified drives. The parameter passed is a 16 bit vector of drives to be reset; the least significant bit is drive A:.

To maintain compatibility with MP/M, CP/M returns a zero value in register A.

Function 40: Write Random With Zero Fill

```
Entry Parameters:
     Register    C:  28H
     Register pair DE:  FCB Address

Returned Value:
     Register    A:  Return Code
```

The "Write Random With Zero Fill" operation is similar to Function 34, with the exception that a previously unallocated block is filled with zeros before the data is written.

CHAPTER 3

DEBUGGING YOUR PROGRAMS

This chapter introduces you to some of the debugging facilities available on the 480Z and 380Z. The first section describes how to patch programs using the Front Panel, the second one explains how to do this using the CP/M DDT utility.

PATCHING PROGRAMS USING THE FRONT PANEL

The debugging features provided by the Front Panel include the display of the contents of some of the Z80 registers, a set of commands for displaying and altering the contents of memory and a command for stepping through a program one instruction at a time. A full description of all these features is given in the Firmware Reference Manual. A more detailed description of how to use the Front Panel is given in the 380Z/480Z Machine Language Programmers Guide.

The current section describes how you can use the Front Panel for one of the most common operations of the debugging process: patching a program.

When the keyboard is being polled for input, you can usually type CTRL F to enter the Front Panel. An automatic entry to the Front Panel occurs whenever the processor executes a break point instruction (code FF hex).

If you want to create a "personal version" of some standard item of software, for example by altering the default value of some of the parameters, then this can be done by "patching" the program using the Front Panel. As an example, to change the initial line length of the TXED formatter option to 72, the procedure would be as follows:

	Commands input	Comments
1	TXED	Load TXED
2	<CTRL/F>	Enter front panel (type "Y" in response to the prompt)
3	M 106 <RETURN>	The prompt ">" will be displayed
4	I	Point to formatter default value table (first byte is initial line length)
5	48 <RETURN>	72 in hex
6	K	Return to TXED
7	EX <ESC><ESC>	Return to CP/M
8	SAVE n EDIT.COM <RETURN>	

Please refer to the TXED Release Note for the exact value of n in step 8.

Further examples can often be found in the documentation for the program concerned.

NOTES:

1 Before patching an item of software make sure you have a back-up copy of the original program

2 When you have patched an item of software give it a NEW name so you can distinguish it from the original program

3 In order to patch an item of software successfully, you will need up-to-date documentation for the program concerned

PATCHING USING THE CP/M DDT UTILITY

If you want to make substantial additions to an existing program then you can use the CP/M DDT Utility Program to "patch" the program. The material below describes some of the facilities provided by the CP/M DDT command.

This section does not describe all of the DDT commands since many of these duplicate features of the Front Panel, which is generally more powerful. If you mistype one of the commands described it may produce a bewildering response if it is one of those recognized by DDT but not described below.

You should note that if a breakpoint instruction (code 0FFH) is executed while under the control of DDT, a DDT breakpoint will be executed and the Front Panel will NOT be entered.

The following procedure is recommended:

1 Load the module(s) to be tested into memory using the DDT I and R commands (described below)

2 Type <CTRL/C> to exit from DDT

3 Type <CTRL/F> to enter the Front Panel if debugging is necessary

4 Save the program with the SAVE command if required

Initiating DDT

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. It is, however, suggested that DDT is <u>not</u> used for debugging purposes as the Front Panel is, in many ways, superior. DDT is initiated by typing one of the following commands at the CP/M Console Command level:

```
DDT
DDT filename.HEX
DDT filename.COM
```

where "filename" is the name of the program to be loaded. In these cases, the DDT program is brought into main memory in the place of the normal CP/M Console Command Processor; thus, it sits directly below the Basic Disc Operating System portion of CP/M. The BDOS starting address, which is located in the address field of the JP instruction at location 0005H, is altered to reflect the reduced Transient Program Area size.

The second and third forms of the DDT command shown above perform the same actions as the first, except that there is a subsequent automatic load of the specified HEX or COM file. The action is similar to the sequence of commands:

```
DDT
Ifilename.HEX or Ifilename.COM
R
```

where the I and R commands set up and read the specified program to be tested (see the explanation of the I and R commands below for exact details).

Operating DDT

Following the sign-on message, DDT prompts the operator with the character "-" and waits for input commands from the console. The operator can type any of several single character commands, terminated by a carriage RETURN, to execute the command. Each command can be up to 32 characters in length (an automatic carriage RETURN is inserted as the 33rd character), where the first character determines the command type. The only two commands normally used are:

I	set up a standard input file control block
R	read program for subsequent testing

Execution of DDT may be terminated at any point by typing <CTRL/C>.

You can save the current contents of memory in a file by using a SAVE command of the form:

```
SAVE n filename.COM
```

where n is the number of pages (256 byte blocks) to be saved on disc (see below).

Remember that when SAVE is used to save a memory image it does not save the machine state. Thus, the program must be restarted from the beginning when you want to use it or when you want to make further tests.

DDT Commands

Two individual commands are described in some detail. In each case, the operator must wait for the prompt character (-) before entering the command.

The I (Input) Command

The I command allows the user to insert a file name into the CP/M default file control block created by CP/M for the use of transient programs. The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. This file name is also used by DDT for reading additional files.

The forms of the I command are

```
        Ifilename
```

or:

```
        Ifilename.filetype
```

Subsequent R commands can be used to read the file (see the R command for further details). The command also allows for any input command line to be set up for the program under test as if it had been passed by the CCP; however, its length is limited to 32 bytes.

The R(Read) Command

The R command is used in conjunction with the I command to read files from the disc into the transient program area in preparation for the debug run. The forms are:

```
        R
        Rb
```

where "b" is an optional bias address which is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 000H through 0FFH (i.e., the first page of memory), nor should it overwrite CP/M, the relocated DDT, or COS workspace. If "b" is omitted, then b = 0000 is assumed. The R command requires a previous I command, specifying the name of a file. The load address for each record is obtained from each individual record of a .HEX file, while an assumed load addess of 100H is taken for files with any other file extension name.

Any number of R commands can be issued following the I command to re-read the program (assuming the default file control block has not been destroyed). Further, any file specified with the filetype "HEX" is assumed to contain machine code in Intel hex format (created, for example, by the ZASM assembler), and all others are assumed to contain machine code in pure binary form (produced, for example, by the SAVE command).

Whenever the R command is issued, DDT responds either with the error indicator "?" (which means that the file cannot be opened, or a checksum error has occurred in a .HEX file), or with a load message taking the form:

```
NEXT PC
nnnn pppp
```

where "nnnn" is the next address following the loaded program and "pppp" is the assumed value of the program counter (either 100H for COM files or taken from the last record if a HEX file is specified).

NOTE:

1 All numerical values input to DDT must be expressed as hexadecimal numbers

2 All numerical values output by DDT are expressed in hexadecimal

3 The DDT loading operation can be used to determine the length in blocks of a program for use in a SAVE Command. For example, enter "A" and at the prompt > enter

```
DDT TEST. COM  <RETURN>
```

The following text (or something similar) will be displayed:

```
NEXT  PC
1D00  0100
```

where "1D00" hex is one greater than the last address in the program; to convert this value to the number of blocks used by the program, use the following algorithm:

1 Subtract 1

2 Now take the first two digits and convert them to decimal. The result is the block size of the program

In this case we have:

```
D00-1=1CFF
```

1C Converts to 28 decimal so we can save a new copy of TEST by entering:

```
<CTRL/C>

SAVE 28 NEW.COM
```

CHAPTER 4

CP/NET BDOS FUNCTIONS

CP/NET is a network operating system that enables microcomputers to access common resources via a network. Programs may be designed specifically for use under CP/NET, or they may have originally been designed for use under CP/M and need changing to run under CP/NET. In both cases, this chapter tells you how to write such programs.

The first section is an introduction to network programming; it describes the capabilities of a network and the various parts of the operating system. The second section tells you how to modify CP/M programs to run under CP/NET and you are then shown how to take advantage of the special facilities of CP/NET. Finally, each CP/NET BDOS function is described in detail.

Before you read on, a final point must be made. This chapter describes the use of local discs on network stations but they may not be available at the time you read it.

INTRODUCTION

In addition to the standard CP/M facilities, CP/NET allows you to do the following:

1 Make use of the scarce, expensive devices which are shared by all work stations on the system, for example, discs and printers

2 Run a program from a work station without knowledge of programs running from other stations

3 Implement an electronic mail system which allows stations to send messages to one another via the server disc system

The standard CP/NET configuration is shown in Figure 4.1. MP/M is an operating system which runs the disc and printer server, CP/NET is the bridge between the server and a number of CP/NET stations.

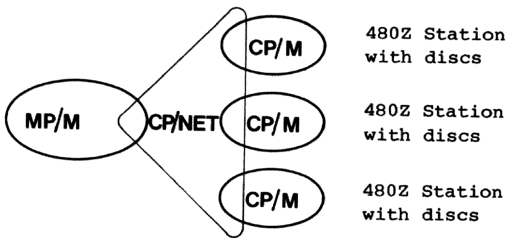


Figure 4.1 Standard CP/NET configuration with local discs

The stations executing CP/M have access to the public resources of the server and to their own local, private resources, which cannot be accessed from the network. This configuration permits the server's resources to be shared amongst stations yet guarantees the security of a station's resources.

In addition to the arrangement described above, it is possible to access CP/NET if your station microcomputer lacks disc resources and is therefore unable to run CP/M; this is shown in Figure 4.2. Here, CP/NOS provides a "virtual" CP/M 2.X system to the station which can consist of no more than a processor, memory and an interface to the network. Thus, a 480Z with sufficient RAM can execute CP/M programs, performing its computing locally and relying upon the network to provide all disc, printer and other I/O facilities.

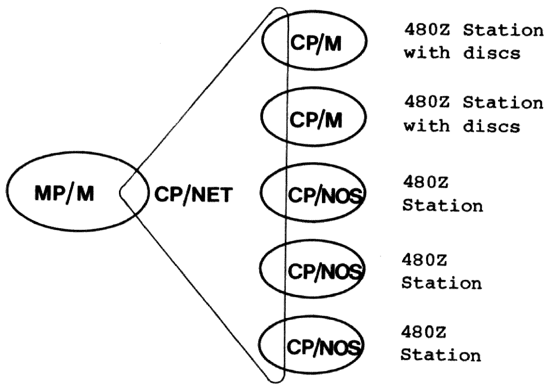


Figure 4.2 Configuration with few local discs

CP/NOS consists of the following:

1 A bootstrap loader which can be placed into ROM or PROM

2 A skeletal CP/M containing only the console and printer functions

3 The logical and physical portions of the CP/NET station

How a station works

The CP/NET station software runs under an unmodified CP/M version 2 operating system. It consists of three modules in addition to the BDOS and BIOS; these are shown in Figure 4.3 and are listed below:

1 The Network Disc Operating System, or NDOS

2 The Station Network I/O System, or SNIOS

3 A replacement for the normal CP/M CCP

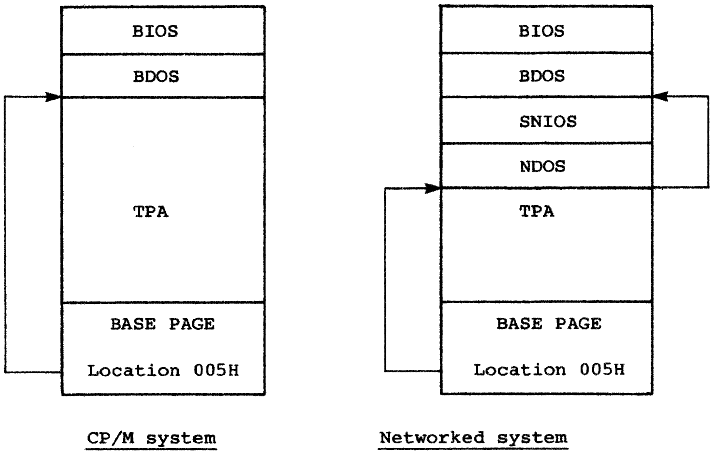


Figure 4.3 The structures of a CP/M system and a networked system

The NDOS determines whether devices referenced by BDOS calls are local to the station or are located on a remote system across the network. If a referenced device is remote, the NDOS prepares messages to be sent across the network and controls their transmission. It also reformats the result received from the network into a form usable by the calling application program.

The SNIOS performs primitive operations which allow the NDOS to send and receive messages across a network. It also provides a number of housekeeping functions for the NDOS. It performs a similar role to the BIOS in customizing the operating system to the hardware of your computer.

When your program performs a BDOS Function call, via location 005H, the

NDOS is entered instead of the BDOS.

MODIFYING PROGRAMS TO RUN UNDER CP/NET

Before converting a program to run under CP/NET you should first ensure that it will run on the 380Z or 480Z under CP/M; the actions needed are described in Chapter 2. Only then should you try to get it working under CP/NET.

All of the CP/M 2.2 functions have corresponding functions under CP/NET. However, there are special conditions relating to the use of discs and printers and these are described below.

First of all, you should use Function 12, "Return version number", to find out if your program is running under CP/NET. If you do not use it and a CP/NET program is run under CP/M, it will fail if any special CP/NET functions are called.

Function 12 returns a two-byte value in register HL. The low-order byte contains the release number and this is set to 22 hexadecimal if CP/NET is running under CP/M version 2.2. The high-order byte specifies the operating system type as given below:

High-order byte	Operating system
0	CP/M
1	MP/M II
2	CP/NET or CP/NOS

The next area you need to look at is file handling. When you make a file under CP/M it is good practice to first open it; if an error is returned, the file can then be assumed not to exist. When running under CP/NET, however, this check will not work: if the file entry is not found in your current directory, the directory of user 0 will be searched for a system file of the same name and, if this exists, it will be opened. To get around this problem, you could check the mode in which the file is opened; if you try to open a file in write mode and it is opened in read mode, you can assume that the file opened is in the directory of user 0.

In summary, if a file is opened in locked or read-only mode from a non-zero user number, the following actions are taken:

1 If the file exists in the same user number, the file is opened

2 If the file does not exist in the same user number, user 0 is searched. If the file exists on user 0, and it is a system file, it is opened just as though the file existed under the other user number

3 If the file exists on user zero as a system file, but it is also a Read-Only file, it will be automatically opened in Read-Only mode

The open and make file functions differ under CP/NET; they return a two-byte value called the file ID in bytes 33 and 34 of the opened FCB. The file ID is needed when performing record locking functions (described in the last section of this chapter).

You should always close files when running under CP/NET and when your program doesn't need a file any more, you should close it as soon as possible, to allow other programs access to it.

When your program outputs to a network printer, the output is spooled and will not be printed until either the program finishes or sends the code FFH to the printer. Your programs should not send FFH to a local printer, neither should they send <CTRL/Z> to the network printer. You can determine if the printer attached to your station is networked by accessing the station configuration table (see this chapter under the heading "Device mapping across the network").

USING THE SPECIAL FACILITIES OF CP/NET

There are a number of special facilities that you can use in programs running under CP/NET. These are summarized below and described in more detail in subsequent sections.

The first of these special facilities is improved error handling on networked devices: using Function 45, "Set BDOS error mode", you can define the way that errors will be handled. You might want them displayed on the screen and/or returned to your program. You might also want to make use of the extended error facility; this gives you more detailed error information than you would have been given under CP/M.

The network contains a configuration table for each station and one for the server. These describe the mapping of devices across the network and your programs can dynamically modify this mapping, if required.

Applications which access networked drives use the MP/M II file system to perform file operations and many of these operations have slightly different meanings than they do under CP/M. Due to the need to prevent several users writing to a file at the same time, a file locking mechanism is necessary. By setting the high-order bits of an FCB filename a file can be opened (or made) in locked mode, unlocked mode or read-only mode. One other point is important when dealing with files: you should always close a file when you have finished with it, even read-only files.

Error handling under CP/NET

CP/NET function calls return specific values in the CPU registers. These values can be pointers to data objects, bit vectors specifying drive status, directory codes, or success/error conditions. Directory, success and error codes are returned in register A; pointers and bit vectors are returned in register HL. Register A is always equal to register L and register B is equal to register H for all CP/NET return codes.

When a CP/NET station performs a local file operation, the function parameters pass untouched to the CP/M BDOS. The BDOS checks these parameters for validity and calls the BIOS to perform physical I/O functions. Two types of error can arise from these local operations:

1 First, the BDOS can detect certain logical problems with a file function and return a logical error. If it does, an error code is returned in register A but the calling application program is allowed to continue

2 Secondly, a physical error is returned when the BIOS is unable to successfully perform a physical operation requested by the BDOS. When the BDOS is presented with a physical error, it prints the following message on the console:

```
BDOS Err on <x>:
<error message>
```

where <x> is the drive referenced when the error occurred and <error message> is one of the four following errors:

(a) Bad Sector

(b) Select

(c) File R/O

(d) R/O

After the physical error message is printed, the BDOS waits for you to respond to the error with one of two actions. Pressing <CTRL/C> causes the BDOS to perform a warm boot, aborting the program. Pressing any other key causes the BDOS to ignore the physical error and continue as if it had not occurred.

When an application references a networked device, the MP/M II server performs the actual file operation and returns a message defining whether the operation was successful or not. Unlike the local case, the station has only indirect knowledge of any error status. Direct physical error indications are impossible to obtain because a station has no contact with the MP/M II input/output coding. Instead, if an error occurs, MP/M II returns a message showing that an error occurred and indicating the type of error it was.

When referencing a remote device, the two types of errors possible under CP/NET are <u>logical errors</u> and <u>extended errors.</u>

Like logical errors under local CP/M, logical network errors define non-fatal error conditions such as reading past the end of a file or attempting to open a nonexistent file. Some serious error conditions are returned as logical errors for functions that expect to process their own errors. These functions are as follows:

Function No.	Function
20	Read Sequential
21	Write Sequential
33	Read Random
34	Write Random
40	Write Random with Zero Fill
42	Lock Record
43	Unlock Record

Errors for these functions are returned in register A, so the condition code upon return to the application program looks exactly as it does under local CP/M.

Some of the following codes can be returned in register A for each of the preceding functions:

Code	Meaning
00	Function successful
01	Reading unwritten data or no directory space available
02	No available data block (disc full)
03	Cannot close current extent
04	Seek to unwritten extent
05	No directory space available
06	Random record greater than 3FFFF
08	Record locked by another process
09	Invalid FCB
0A	FCB checksum error
0B	File verify error
0C	Record lock limit exceeded
0D	Invalid file ID
0E	No room in System Lock List

Extended errors indicate that a potentially fatal condition has occurred during the execution of an MP/M II function. The condition can be a physical error, similar to the physical errors that can occur under CP/M. It can also be an error produced by the file system, indicating that the specified operation violates the integrity of the file system.

There are three ways in which errors can be handled; they are as follows:

1 Default Mode

2 Return-Error Mode

3 Return-and-Display-Error Mode

The mode which is in use at a particular time can be defined by you using BDOS Function 45, "Set BDOS error mode". This does not exist under CP/M and because of this, most CP/M applications run in default mode and abort if an extended error occurs.

If the NDOS is in default mode, it prints the following error message:

```
NDOS Err <xx>, Func <yy>
```

where <xx> is the extended error code (in hexadecimal) and <yy> is the function being performed when the error occurred (also in hexadecimal). The NDOS then performs a warm boot, aborting the program.

In return error mode, the NDOS does not display a message or abort the program. Instead, it sets register A to FF and register H to the extended error code; it then returns to the application program.

If an extended error is detected in return-and-display-error mode, the NDOS displays the error message on the console. However, it does not abort the program, setting the registers in the same manner as return-error mode.

The following extended error codes can be returned to the NDOS:

Code	Error
01	Bad sector-permanent disc error
02	Read-only disc
03	Read-only file
04	Drive select error
05	File open by another process in locked mode
06	Close checksum error
07	Password error
08	File already exists
09	Illegal ? in an FCB
0A	Open file limit exceeded
0B	No room in System Lock List
0C	Requester not logged on the server or function not implemented on server
.	
FF	Unspecified physical error

Extended error FF can result from only two special functions: Function 27, "Get Allocation Vector Address" and Function 31, "Get Disc Parameter Address". Because these functions return a pointer in register pair HL, it is not possible to detect a regular extended error. Instead, these functions return an FFFF value in HL if a physical error occurs.

Not all CP/NET functions are capable of returning extended errors. However, extended error 0C can be returned on any function. If an extended error is returned for such a function, the NDOS ignores it. The following functions can result in the performance of a network access but cannot produce an extended error:

Code	Error
1	Console Input
2	Console Output
5	List Output
9	Print String
10	Read Console Buffer
24	Return Login Vector
28	Write Protect Disc
29	Get Read-Only Vector
37	Reset Drive
39	Free Drive
64	Login
66	Send Message on Network
67	Receive Message on Network
70	Set Compatibility Attributes
106	Set Default Password

Any other function can cause a program to abort if an MP/M II extended error occurs, if an unsupported function is passed to the server, or if the server is not logged in.

Device mapping across the network

The mapping of devices across the network is handled via configuration tables. These map logical devices to physical devices and there is one configuration table for each station and one for the server. You can dynamically alter the mapping of devices using Function 69, "Get station configuration table address", and you can look at the server configuration table using Function 71, "Get server configuration table address".

Printer I/O is handled in the following manner: when the BIOS call is made the NDOS traps it. The NDOS examines the configuration table to see if the printer is local to the CP/NET system or networked. If the printer is local, the call is passed through to the BIOS unchanged.

If the printer is networked, however, the NDOS stores the character to be printed in a special buffer, located directly below the station configuration table. When 128 characters are stored, the NDOS sends a List Output logical message to the server upon which the printer is mapped. This buffering process improves system performance because one-character messages that would congest the network communication interfaces need not be sent between each station and server.

A station configuration table has the following format:

Offset in bytes	Purpose
0	STATION STATUS BYTE This has the format shown in Figure 4.4.
1	STATION PROCESSOR ID
2-33	DISC DEVICES This consists of 16 two-byte pairs, one for each drive. If the most-significant bit of the first byte is set, the drive is on the network. The server drive code should be in the least-significant four bits of the first byte. The second byte should contain the server processor ID
34-35	NOT USED
36-37	LIST DEVICE If the most-significant bit of the first byte is set, listing will be output to the network. The server list device number should be in the least significant four bits. The second byte should contain the server processor ID
38	Used by system
39	Used by system
44	List device number of server
45-172	List device buffer

Table 4.1 Station configuration table

The format of the station status byte is shown in Figure 4.4.

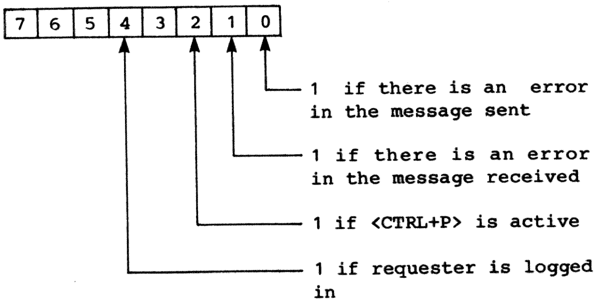

7 6 5 4 3 2 1 0
1 if there is an error in the message sent
1 if there is an error in the message received
1 if <CTRL+P> is active
1 if requester is logged in

Figure 4.4 Format of station status byte

The server configuration table has the following format:

Offset in bytes	Purpose
0	Server temporary file drive
1	Server network status byte
2	Server ID
3	Maximum number of stations permitted on the server
4	Number of stations currently logged in
5-6	Bit vector of stations shown as logged-in in the station ID table

Table 4.2 Server configuration table

This information is similar to that contained in the server configuration table.

Password Protection Under CP/NET

File access by unprivileged users can be limited through password protection for individual files. There are three levels of password protection for files:

1 All access without the password is denied

2 The file can be read without the password, but it cannot be written to

3 The file can be read and written to without the password, but not deleted

You can use the SET utility to assign passwords; this is described in the Research Machines publication:

Network Release 2.1 User's Guide, PN 12262

CP/NET does not support the assignment of passwords across the network. It does, however, allow an application program to send a password across the network when a file is opened. This allows a user on a CP/NET station the most basic form of password support: operation on networked files which have been previously password-protected.

If a read-protected file is opened and no password is specified, an extended error is returned across the network and the calling application aborts. The same error is also returned when an application attempts to write to a write-protected file for which no password was provided when the file was opened. Finally, any attempt to delete, rename, or change the attributes of a delete-protected file without providing a password results in an extended error.

CP/NET also supports Function 106 (Set Default Password). This provides a default password against which all protected files are checked if no password is provided or if the password is incorrect. This function can relieve an application of the necessity to parse passwords constantly into the first eight bytes of the current DMA buffer.

CCP.SPR does not support MP/M II's facility of supplying passwords when you enter a command line. Because of this, you should not password-protect COM files unless a default password utility is provided.

Because CP/M 2.x does not support any kind of file protection, passwords are ignored when referencing files on drives local to a CP/NET station.

TEMPORARY FILENAME TRANSLATION

Many common application programs use temporary files. The names of these files often have the form FILENAME.$$$ or $$$.SUB. When multiple copies of these applications run on different stations logged on to the same server, a number of the temporary files can have the same name, thus causing extended MP/M II errors that abort the application program.

To solve this problem, each station's NDOS recognizes temporary filenames destined for networked drives and implicitly renames them, so the filename an application presents to the operating system is not the one the NDOS presents to the MP/M II file system.

Each occurrence of the string $$$ in the first three bytes of a filename, as well as any filetype of $$$, forms a CP/NET message with a filename or filetype of $<xx>, where <xx> is the ASCII representation of the station ID byte. Because all stations have a unique ID, this modification guarantees the uniqueness of temporary filenames.

The modification is transparent to the calling application program. When the NDOS modifies a filename in a CP/NET message, it converts the filename back to its original form before updating the application's FCB. The only possible change to the FCB is that interface attributes set in the high-order bits of the filename strings modified are reset. This change poses no problems if temporary files are truly temporary. You should treat temporary files like Read-Write files with the DIR attribute; delete them before the application program terminates.

Functions 17 (Search For First Directory Entry) and 18 (Search For Next Directory Entry) do not perform temporary filename translation when referencing a networked drive. If a user creates a file with a temporary filename and then attempts to locate it within his directory, this can be confusing.

For example, suppose that a user working on station 5A enters the command:

```
REN $$$.$$$=BLAH.TMP
```

Suppose, then, the user enters a DIR command. The file previously renamed will appear as

```
$5A.$5A
```

in the directory.

If a temporary file is referenced on a drive that is local to the CP/NET system, the filename passes unmodified to the BDOS. No conversion is necessary, because there is no possibility of conflict.

CP/NET BDOS FUNCTIONS

This section describes CP/NET functions which have no counterpart under CP/M. They are listed below.

Function No	Function	Function No	Function
38	Access drive	66/67	Reserved
39	Free drive	68	Get network status
40/41	Reserved	69	Get station configuration table address
42	Lock record	71	Get server configuration table address
43	Unlock record	72/105	Reserved
44	Reserved	106	Set default password
45	Set BDOS error mode		
46/64	Reserved		
65	Log off		

These functions include MP/M II functions which do not exist under CP/M, as well as a set of dedicated CP/NET functions. All of the functions adhere to exactly the same calling conventions as the rest of the CP/M functions and all follow the same conventions regarding return codes.

Function 38: Access Drive

Entry Parameters:

Register	C:	26H
	DE:	Drive Vector in the form of a bit map

Return Values:

Register	A:	Return Code
	H:	Extended Error

The server maintains a system lock list which contains an entry for every disc drive in the system. If a program sets one of these entries, using the "Access drive" function, the drive concerned will be locked in respect of access from another program.

The "Access Drive" function inserts a dummy open file item in the system lock list for each drive specified in its drive vector (a 16-bit vector in which each possible drive is represented). Bit 0 represents drive A:, bit 1 drive B:, continuing through 15 for drive P:.

If the server's system lock list does not have enough room to hold all the dummy items for all the drives specified, or if the open file limit for the server process is exceeded, none of the items is inserted and Function 38 returns an extended error.

If the NDOS is in return error mode (see Function 45, "Set BDOS error mode"), an error condition on Function 38 causes register A to be set to 0FFH and register H contains one of the following codes:

Code	Meaning
0A	Open File Limit Exceeded
0B	No Room in the System Lock List
0C	Server Not Logged In

Because Function 38 is meaningless to local drives under CP/NET, no call to the local BDOS is made.

Function 39: Free drive

```
Entry Parameters:
     Register  C:   27H
              DE:   Drive Vector in bit map form
```

The "Free Drive" function purges the server's lock list of all items relating to the drives specified. The drive vector is a 16-bit vector in which each possible drive is represented. Bit 0 represents drive A:, bit 1 drive B:, continuing through 15 for drive P:.

Because dummy drive accesses, locked records, and open files are all purged, you should close all important files before issuing a free drive call. Otherwise, a checksum error is returned on the next file access and data might be lost.

The CP/NET CCP calls this function every time a program terminates. This prevents the server process associated with the station from becoming clogged with useless files.

Because the "Free Drive" function is meaningless under CP/M, the operating system ignores entries in the drive vector which specify drives local to the station.

This function has no error return.

Function 42 Lock record

```
Entry Parameters:
     Register  C:   2AH
              DE:   FCB Address

Return Values:
     Register  A:   Return Code
               H:   Extended Error
```

The "Lock record" function gives your program exclusive write access to a specific record of a file opened in unlocked mode. Using it, any number of station processes can simultaneously update a common file.

To lock a record, your program must place the logical record number to be locked in bytes 33 to 35 of the file's FCB. The file ID number must be placed in the first two bytes of the current DMA buffer (this file ID number is a two-byte value which is returned in bytes 33 to 35 of the FCB when the file is opened in unlocked mode). When the "Lock record" function is called, a pointer to the FCB must exist in register pair DE.

The record to be locked must reside within a block which is currently allocated for the file. The lock fails if the record is locked by another process or station (this prevents two processes from simultaneously updating the same record and leaving it in an indeterminate state).

If a file was opened in locked mode, the "Lock record" function always returns successfully but no explicit action is taken because the whole file is locked in the first place.

To use the "Lock record" function, you should follow these steps:

1 Open the file in unlocked mode, then save the file ID which was returned in the random record field of the open FCB

2 When the application needs to update the record, lock the record even before attempting to read it. Reading a record which is locked by another process can result in you leaving the record in an indeterminate state. If an error results because the record is locked by another process, you should repeat this step until the record is locked successfully. When retrying the lock, place a timeout value in case another station has locked the record and then gone off line

3 Read the record

4 Update the record

5 Write the record back

6 Unlock the record

The "Lock record" function returns a zero in register A if successful. Otherwise, it returns one of the following error codes in register A:

01	Reading unwritten data
03	Cannot close current extent to access extent specified
04	Seek to an unwritten extent
06	Random record number greater than 3FFFF
08	Record locked by another process
0A	FCB checksum error
0B	Unlock file verification error
0C	Process record lock limit exceeded
0D	Invalid file ID in the DMA buffer
0E	No room on the system lock list
FF	Extended error

The following extended errors can occur:

01	Permanent error
04	Select error
0C	Requester not logged in to server

The "Lock record" function has no meaning when a drive local to the station is referenced. It returns with register A set to zero.

Function 43 Unlock record

Entry Parameters:

Register	C:	2BH
	DE:	FCB Address

Return Values:

Register	A:	Return Code
	H:	Extended Error

The "Unlock record" function releases a previously locked record, allowing it to be locked and written to by another station. The record to be unlocked must be placed in bytes 33 to 35 of the file's FCB. The file ID must be placed in the first two bytes of the current DMA buffer (the file ID is a two-byte value which is returned in bytes 33 to 35 of the FCB when a file is opened in unlocked mode). Register pair DE must contain a pointer to the FCB.

The "Unlock record" function returns successfully if one of the following occurs:

1 The file was opened in locked mode

2 The record specified is already unlocked

3 The record is locked by another process

In all these cases no action is performed.

You should not unlock a record until the station's application program has finished updating the locked record and has written it back out to the file. Otherwise, another process might inadvertently destroy the updated information.

The "Unlock record" function returns a zero in register A if successful. Otherwise, the function returns one of the following error codes in register A:

Code	Meaning
01	Reading unwritten data
03	Cannot close current extent to access extent specified
04	Seek to an unwritten extent
06	Random record number greater than 3FFFF
0A	FCB checksum error
0B	Unlock file verification error
0D	Invalid file ID in the DMA buffer
FF	Extended error

The following extended errors can occur:

01	Permanent error
04	Select error
0C	Server not logged in

The "Unlock record" function is meaningless when it references a station's local drive; it returns a zero in register A.

Function 45: Set BDOS error mode

Entry Parameters:

Register	C:	2DH
	E:	Error Mode

CP/M returns a very limited amount of error information. However, when running under CP/NET you can ask the system to generate further information by using Function 45, "Set BDOS error mode".

When this function has been called, the NDOS is provided with the following options:

1 Aborting on extended errors

2 Returning the extended error to the calling application for handling

3 Returning the error to the application and displaying it on the console

All station application programs are initially loaded in a default environment. This causes the NDOS to abort on extended errors and to display the extended error code. You should use Function 45 to change this default mode to a mode which depends upon the contents of register E. The values you can specify are shown in Table 4.3.

Table 4.3 BDOS Error Modes

Register E	Explanation
0FFH	Return-Error Mode. BDOS returns extended errors coming from the network to the application program. Register A is set to 0FFH, and register H contains the extended error code. No error message is displayed on the console.
0FEH	Return-and-Display Mode. BDOS returns the extended error in the same manner as in Return-Error Mode, but also displays an extended error message.
Any Other Value	Default Mode.

Function 45 is not implemented across the network. The NDOS maintains its own internal error mode flag and acts upon returning network messages according to that flag.

The "Set BDOS error mode" function has no effect on physical errors returned by the station's local BIOS. These errors always display an error message, then they give you the option of aborting the application program or continuing.

Function: 65 Logoff

Entry Parameters:

Register	C:	41H
	E:	Server ID

Return Values:

Register	A:	Return Code
	H:	Extended Error

The "Logoff" function completes a session and breaks the logical binding between the server specified in register E and the calling station. Once a Logoff has been performed, the server process is free to begin a session with another station.

Function 65 returns a 0 if successful. It returns an extended error 0C, station not logged on to server, if unsuccessful.

Function 68: Get network status

Entry Parameters:

Register	C:	44H

Return Values:

Register	A:	Network Status Byte

The "Get network status" function returns the station configuration table's network status byte in register A. It also resets any error conditions in the status byte.

For a description of the fields contained in the station status byte, see Figure 4.4.

Function: 69: Get station configuration table address

Entry Parameters:

Register	C:	45H

Return Values:

Register	HL:	Table Address

The "Get Station Configuration Table Address" function returns the address of the station configuration table maintained in the SNIOS. Using this function, an application can dynamically modify the mappings of devices across the network. The utilities NETWORK and LOCAL use Function 69 to accomplish this kind of modification.

For a description of the fields in the station configuration table, see Table 4.1.

Function 71: Get server configuration table address

Entry Parameters:

Register	C:	47H
	E:	Server ID

Return Values:

Register	HL:	Server Configuration Table Address

The "Get server configuration table address" function returns a pointer to a copy of parts of the specified server's configuration table. The ID of the server to be examined is passed in register E prior to calling Function 71 and a pointer to the received information is returned in register pair HL.

The data structure addressed by HL has the following format:

Byte	Purpose
00-00	Server Temporary File Drive
01-01	Server Network Status Byte
02-02	Server ID
03-03	Maximum Number of Requesters Permitted on the Server
04-04	Number of Requesters Currently Logged In
05-06	Bit Vector of Requesters Logged In in the Requester ID Table
07-16	Requester ID Table

This information is identical to that contained in the server configuration table, except that a byte containing the server's temporary file drive has been added to the front of the table.

Function 71 can determine whether other stations are logged in to a server. The temporary file drive can be used when an application wants to leave a file on a server but does not know the names, capacity, or type of the server's disc drives. The MAIL utility makes frequent use of Function 71.

The server configuration table is returned across the network into a special buffer in the NDOS. If more than one call is to be made to Function 71 and the calls reference a different server each time, the buffer is overwritten by each successive call. If an application must examine more than one server configuration table at a time, the table must be copied down into a buffer defined by the application.

If Function 71 passes a server ID to which the calling station is not logged on, an extended error 0C (Station Not Logged In) is returned.

Function 106: Set default password

```
Entry Parameters:
     Register  C:   46H
              DE:   Password Address
```

The "Set default password" function allows an application to specify a password that is checked if an incorrect password is presented during an "Open file" function call. If a file is password protected, MP/M II first checks for a password in the current DMA buffer. If no match is found, MP/M II then checks the default password set by Function 106. If it finds a match, it allows the requested operation to succeed. Otherwise, it returns an error.

When Function 106 is performed on a station, the station's NDOS attempts to set the default password on every server to which a drive is networked by that station. Since Function 106 has no error return, extended station-not-logged-in errors are ignored.

Each server process uses an MP/M II default password slot, starting with console 0 and using as many slots as there are stations supported.

The default password set by Function 106 persists until another default password is set.

CHAPTER 5

MORE ADVANCED USE OF CP/M

There are a number of things which you may want to do to your system to personalize it: you might want to get more information from the BIOS disc handling routines or add a new device. The next section tells you how you can do this by taking, as an example, the addition of a new device. Following this is a section which describes the BIOS internal routines in detail. The two final sections describe the tables which define the disc system in use and the layout of page zero of memory.

ADDING A DEVICE HANDLER

If you add an unsupported device to your system you will need to write a "device handler" to handle the interface between CP/M and that device. Once you have done that you will need to build it into CP/M. There are two stages to this:

1 You need to rebuild CP/M to run in a different size of memory; this will ensure that your driver can run in a protected area of memory

2 You then need to connect your handler to CP/M

These aspects are covered below.

Running CP/M in a different area of memory

You can reduce the size of the TPA by running MOVCPM. For example, the following call will reduce the size of a 56Kb system to 55Kb:

```
MOVCPM 55 *
```

The new operating system will be built in a buffer and will not be executed. You can then use SYSGEN to copy the operating system to another disc. This procedure is described in the following manual:

CP/M Operating System Version 2.2D Users Guide, PN 11901

Now, when you load this new disc and press the "B" command, the cold boot loader will load your new system.

The above sequence describes the basic principles; however, you need to add your device handler and connect it up before the new system is written to disc. You can do this by stopping SYSGEN when the new operating system is in the buffer, then patching it using the Front Panel.

Connecting your device handler to CP/M

The BIOS uses a "jump vector" to point to its device handling routines. The jump vector is a sequence of 17 jump instructions that send program control to the individual BIOS subroutines. The BIOS subroutines may be empty for certain functions (i.e., they may contain a single RET operation) during reconfiguration of CP/M, but the entries must be present in the jump vector.

The jump vector takes the form shown in Table 5.1.

Offset from start of BIOS (in HEX)	Instruction	Purpose
0	JMP BOOT	;ARRIVE HERE FROM COLD ;START LOAD
3	JMP WBOOT	;ARRIVE HERE FOR WARM START
6	JMP CONST	;CHECK FOR CONSOLE CHAR READY
9	JMP CONIN	;READ CONSOLE CHARACTER IN
0C	JMP CONOUT	;WRITE CONSOLE CHARACTER OUT
0F	JMP LIST	;WRITE LISTING CHARACTER OUT
12	JMP PUNCH	;WRITE CHARACTER TO PUNCH DEVICE
15	JMP READER	;READ READER DEVICE
18	JMP HOME	;MOVE TO TRACK 00 ON ;SELECTED DISK
1B	JMP SELDSK	;SELECT DISK DRIVE
1E	JMP SETTRK	;SET TRACK NUMBER
21	JMP SETSEC	;SET SECTOR NUMBER
24	JMP SETDMA	;SET DMA ADDRESS
27	JMP READ	;READ SELECTED SECTOR
2A	JMP WRITE	;WRITE SELECTED SECTOR
2D	JMP LISTST	;RETURN LIST STATUS
30	JMP SECTRAN	;SECTOR TRANSLATE SUBROUTINE

Table 5.1 The BIOS jump vector

The address of the warm start vector is held in locations 0001H and 0002H. In your programs you should define this point and access the jump instructions in terms of offsets from it.

There are three major divisions in the jump table: the system (re)initialization, which results from calls on BOOT and WBOOT; simple character I/O performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST; and disc I/O performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTRAN.

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII CTRL-Z (1AH). Peripheral devices are seen by CP/M as "logical" devices and are assigned to physical devices within the BIOS.

To operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines (LISTST may be used by PIP, but is not used by the BDOS).

The characteristics of each device are shown in Table 5.2.

Device	Characteristics
CONSOLE	This is the keyboard and screen on the 480Z and 380Z. It is accessed through CONST, CONIN, and CONOUT
LIST	The principal listing device, if it exists on your system, is usually a printer
PUNCH	This is a "null" device
READER	This is mapped to the SIO-4 port

Table 5.2 Device characteristics

A single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously.

Disc I/O is always performed through a sequence of calls on the various disc access subroutines that set up the disc number to be accessed, the track and sector on a particular disc and the direct memory access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation. There is often a single call to SELDSK to select a disc drive, followed by a number of read or write operations to the selected disc before selecting another drive for subsequent operations. Similarly, there may be a single call to set the DMA address, followed by several calls that read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

The READ and WRITE routines perform several retries (10 is standard) before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it will report the error to your program.

The exact responsibilities of each entry point subroutine are given in the next section.

BIOS entry point subroutines

The jump vector contains a number of jump instructions which point to the subroutines which handle each device. These subroutines are described below.

BOOT — The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a sign-on message. The various system parameters that are set by the WBOOT entry point must be initialized, and control is transferred to the CCP for further processing. Note that register C must be set to zero to select drive A.

WBOOT — The WBOOT entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H, or when the CPU is reset from the Front Panel. The CP/M system is loaded from the first two tracks of drive A up to, but not including, the BIOS System parameters are initialized as shown below:

location 0,1,2	Set to JMP WBOOT for warm starts
location 4 (used only by CCP)	High nibble = current user no; low nibble = current drive
location 5,6,7	Set to JMP BDOS, which is the primary entry point to CP/M for transient programs

(You should refer to the last section in this chapter for complete details of page zero use.) Upon completion of the initialization, the WBOOT program branches to the CCP to (re)start the system. Upon entry to the CCP, register C is set to the drive to select after system initialization.

CONST — Samples the status of the currently assigned console device and returns 0FFH in register A if a character is ready to read and 00H in register A if no console characters are ready.

CONIN — The next console character is read into register A, and the parity bit is set (high order bit) to zero. If no console character is ready, this routine waits until a character is typed before returning.

CONOUT	Sends the character from register C to the console output device. The character is in ASCII, with high order parity bit set to zero.
LIST	Sends the character from register C to the punch device. The character is in ASCII with zero parity.
PUNCH	Sends the character from register C to the punch device. The character is in ASCII with zero parity.
READER	Reads the next character from the reader device into register A with zero parity (high order bit must be zero); an end-of-file condition is reported by returning an ASCII CTRL/Z(1AH).
HOME	Moves the disc head of the currently selected disc (initially disc A) to the track 00 position
SELDSK	Selects the disc drive given by register C for further operations, where register C contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disc select, SELDSK returns, in HL, the base address of a 16-byte area, called the Disk Parameter Header, described in the next section. For standard floppy disc drives, the contents of the header and associated tables do not change. If there is an attempt to select a nonexistent drive, SELDSK returns HL=0000H as an error indicator.
SETTRK	Register BC contains the track number for subsequent disc accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point. Register BC can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disc drives and 0-65535 for non-standard disk subsystems.
SETSEC	Register BC contains the sector number (1 through 26) for subsequent disc accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point.
SETDMA	Register BC contains the DMA (disc memory access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, all subsequent read operations read their data into 80H through 0FFH and all subsequent write operations get their data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H.
READ	Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA address has been specified, the READ subroutine attempts to read one sector

based upon these parameters and returns the following error codes in register A:

0	no errors occurred
1	non-recoverable error condition occurred

Currently, CP/M responds only to a zero or non-zero value as the return code. That is, if the value in register A is 0, CP/M assumes that the disk operation was completed properly. If an error occurs, however, BIOS attempts at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of pressing RETURN to ignore the error, or CTRL/C to abort.

WRITE	Writes the data from the currently selected DMA address to the currently selected drive, track, and sector. The error codes given in the READ command are returned in register A, with error recovery attempts as described above.
LISTST	Returns the ready status of the list device. The value 00 is returned in A if the list device is not ready to accept a character and 0FFH if a character can be sent to the printer.
SECTRAN	Performs logical-to-physical sector translation to improve the overall response of CP/M. In general, SECTRAN receives a logical sector number relative to zero in BC, and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL.

DISC PARAMETER TABLES

Tables are included in the BIOS to describe the particular characteristics of the disc subsystem used with CP/M. This section describes the elements of these tables.

In general, each disc drive has an associated (16-byte) disc parameter header that contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disc parameter header for each drive is shown below.

Disc Parameter Header

XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16b	16b	16b	16b	16b	16b	16b	16b

where each element is a word (16-bit) value. The meaning of each Disc Parameter Header (DPH) element is as follows:

XLT	Address of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e., the physical and logical sector numbers are the same)
0000	Scratchpad values for use within the BDOS (initial value is unimportant)
DIRBUF	Address of a 128-byte scratchpad area for directory operations within BDOS. All DPHs address the same scratchpad area
DPB	Address of a disc parameter block for this drive. Drives with identical disc characteristics address the same disc parameter block
CSV	Address of a scratchpad area used by software check for changed discs. The address is different for each DPH
ALV	Address of a scratchpad area used by the BDOS to keep disc storage allocation information. This address is different for each DPH

The disc parameter block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPHs, takes the general form

SPT	BSH	BLM	EXM	DSM	DRM	AL0	AL1	CKS	OFF
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b

where each is a byte or word value, as shown by the 8b or 16b indicator below the field. The fields are shown in Table 5.3.

Field	Purpose
SPT	The total number of sectors per track
BSH	The data allocation block shift factor, determined by the data block allocation size
BLM	The data allocation block mask (2[BSH-1])
EXM	The extent mask, determined by the data block allocation size and the number of disc blocks
DSM	Determines the total storage capacity of the disc drive
DRM	Determines the total number of directory entries that can be stored on this drive (AL0,AL1 determine reserved directory blocks)
CKS	The size of the directory check vector
OFF	The number of reserved tracks at the beginning of the (logical) disc

Table 5.3 Fields of disc parameter block

The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the DPB. Assuming that a value has been given for BLS, the values of BSH and BLM are shown in the table below.

BLS	BSH	BLM
1024	3	7
2048	4	15
4096	5	31
8192	6	63
16384	7	127

where all values are in decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255. For DSM<256 the value of EXM is given by:

BLS	EXM
1024	0
2048	1
4096	3
8192	7
16384	15

For DSM>255 the value of EXM is given by:

BLS	EXM
1024	N/A
2048	0
4096	1
8192	3
16384	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disc, not counting the reserved operating system tracks.

The DRM entry is one less than the total number of directory entries. The values of AL0 and AL1 are determined by DRM. The values AL0 and AL1 can together be considered a string of 16-bits, as shown below.

AL0								AL1							
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15

where position 00 corresponds to the high order bit of the byte labeled AL0 and 15 corresponds to the low order bit of the byte labeled AL1. Each bit position reserves a data block for a number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, resulting in the tabulation below.

BLS	Directory Entries
1024	32 times # bits
2048	64 times # bits
4096	128 times # bits
8192	256 times # bits
16384	512 times # bits

Thus, if DRM = 127 (128 directory entries) and BLS = 1024, there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disc drive media is removable, then CKS = (DRM+1)/4, where DRM is the last directory entry number. If the media are fixed, then set CKS = 0 (no directory records are checked in this case.)

Finally, the OFF field determines the number of tracks that are skipped at the beginning of the physical disc. This value is automatically added whenever SETTRK is called and can be used as a mechanism for skipping reserved operating system tracks or for partitioning a large disc into smaller segmented sections.

Returning to the DPH for a particular drive, the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If CKS = (DRM+1)/4, (DRM+1)/4 bytes are reserved for directory check use. If CKS = 0, no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disc and is computed as (DSM/8)+1.

RESERVED LOCATIONS IN PAGE ZERO

Main memory page zero, between locations 00H and 0FFH, contains several segments of code and data that are used during CP/M processing. The code and data areas are given in Table 5.4 for reference:

This information is set up for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient.

Due to the system structure, it is a non-trivial task to run programs from location 0H. If you do this, you must take all responsibility for the results. As an added disadvantage, you will not be able to use the firmware facilities.

Locations from to	Contents
0000H-0002H	Contains a jump instruction to the warm start entry point. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel
0003H-0003H	Contains the Intel IOBYTE (not used)
0004H-0004H	Current default drive number (0=A,...,15=P) and user number
0005H-0007H	Contains a jump instruction to the BDOS and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in Chapter 2, and LD HL,(0006H) brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M (assuming the CCP is being overlaid). The DDT program will change the address field to reflect the reduced memory size in debug mode.
0008H-002FH	(Interrupt locations 8 through 28; not used) (Locations 0028-002FH and 0010-0015H used by) (COS/ROS)
0030H-0037H	(Used by COS/ROS)
0038H-003AH	(Used by the Front Panel)
003BH-003FH	(Not currently used; reserved)
0040H-004FH	A 16-byte area reserved for scratch by BIOS
0050H-005BH	(Not currently used; reserved)
005CH-007CH	Default file control block produced for a transient program by the CCP
007DH-007FH	Optional default random record position.
0080H-00FFH	Default 128-byte disc buffer (also filled with the command line when a transient is loaded under the CCP)

Table 5.4 Reserved locations in page zero

APPENDIX A

A SAMPLE FILE-TO-FILE COPY PROGRAM

The program on the following pages provides a relatively simple example of sequential file operations. The program source file is created as COPY.ZASM using the CP/M TXED program and then assembled using ZASM, resulting in a HEX file. The LOAD program is used to produce a COPY.COM file, which executes directly under the CCP. The program begins by setting the stack pointer to a local area and proceeds to move the second name from the default area at 006CH to a 33-byte file control block called DFCB. The DFCB is then prepared for file operations by clearing the current record field. At this point, the source and destination FCBs are ready for processing, since the SFCB at 005CH is properly set up by the CCP upon entry to the COPY program. That is, the first name is placed into the default FCB, with the proper fields zeroed, including the current record field at 007CH. The program continues by opening the source file, deleting any existing destination file and creating a new destination file. If a ll this is successful, the program loops at the label COPY until each record has been 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.

You should note that there are several simplifications in this particular program. First, there are no checks for invalid file names 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. A check should also be made to ensure that the file names have, in fact, been included (check locations 005DH and 006DH for nonblank ASCII characters). Finally, a check should be made to ensure that the source and destination file names are different. An improvement in speed could be obtained by buffering more data on each read operation. One could, for example, determine the size of memory by fetching FBASE from location 0006H and using the entire remaining portion of memory for a data buffer. In this case, the programmer simply resets the DMA address to the next successive 128-byte area before each read. Upon writing to the destination file, the DMA address is reset to the beginning of the buffer and incremented by 128 bytes to the end as each record is transferred to the destination file.

```
*H COPY Version 2.0A

   ************************************************************************
   *                                                                      *
 ; * COPY                                                 Version 2.1A    *
   *                                                                      *
   * This program copies the contents of a specified file into a          *
 ; * second specified file. The destination file will initially be        *
 ; * deleted if it already exists.                                        *
 ; * COPY is invoked by a command line of the form :                      *
 ; *                                                                      *
 ; *              COPY X:SOURCE.EXT Y:DEST.EXT                            *
 ; *                                                                      *
 ; * with the source and destination files in the order shown.            *
 ; *                                                                      *
 ; * Modified.                                                            *
 ; *     13 Dec 83         to produce COM file directly with ZASM 4.1J    *
 ; *                                                                      *
 ; * Derived from a program supplied by Digital Research.                 *
 ; *                                                                      *
 ; * Assembled using ZASM Version 4.1 J                                   *
 ; ************************************************************************

```

```
                  0024 *H Equates
                  0025boot            equ     0000H          ; used to exit program
 0005 =           0026 bdos            equ     0005H           ; BDOS entry point
 0100 =           0027 tpa             equ     0100H           ; Start of TPA
 005C =           0028 source_filename equ     005CH
 006C =           0029 dest_filename   equ     006CH           ; filenames placed at these
                  0030                                         ; addresses by CCP before entry
 0080 =           0031 file_buffer     equ     0080H
 005C =           0032 source_fcb      equ     source_filename
                  0033
                  0034 ;       BDOS function numbers
                  0035
 0009 =           0036 fn_print_buffer equ     9.
 000F =           0037 fn_open_file    equ     15.
 0010 =           0038 fn_close_file   equ     16.
 0013 =           0039 fn_delete_file  equ     19.
 0014 =           0040 fn_read_file    equ     20.
 0015 =           0041 fn_write_file   equ     21.
 0016 =           0042 fn_create_file  equ     22.
                  0043
                  0044 ;       General equates
                  0045
 0020 =           0046 fcb_length      equ     32.
 0000 =           0047 zero            equ     0
                  0048
 FFFF =           0049 file_not_found_flag     equ     -1
 FFFF =           0050 direct_full_flag        equ     -1
 0000 =           0051 end_of_file_flag        equ     0
 0000 =           0052 disc_full_flag          equ     0
 FFFF =           0053 wrt_prtct_flag          equ     -1
                  0054
```

```
                 0055 *H Main program
                 0056
                 0057         COM
 0100            0058         ORG     tpa
                 0059
                 0060 ;       set up stack
                 0061
 0100 ED73E101   0062
 0104 312402     0063         LD      SP,stack                ; local stack
                 0064
                 0065 ;       move second file name to dest. fcb
                 0066
 0107 011000     0067         LD      BC,fcb_length / 2
 010A 216C00     0068         LD      HL,dest_filename
 010D 11E301     0069         LD      DE,dest_fcb
 0110 EDB0       0070         LDIR                            ; move filename
                 0071
                 0072 ;       name has been removed, set current record to zero
                 0073
 0112 3E00       0074         LD      A,zero
 0114 320302     0075         LD      ( dest_fcb.current_record ),A
                 0076
                 0077 ;       source and destination fcbs ready
                 0078
 0117 115C00     0079         LD      DE,source_filename
 011A CD6A01     0080         CALL    open
                 0081
 011D 118E01     0082         LD      DE,msg_no_source_file
 0120 FEFF       0083         CP      file_not_found_flag     ; does source exist ?
 0122 283C       0084         JR      Z,exit_program          ; no, skip to end
                 0085
                 0086 ;       source file open, prep destination
                 0087
 0124 11E301     0088         LD      DE,dest_fcb
 0127 CD7601     0089         CALL    delete                  ; remove if present
                 0090
 012A 11E301     0091         LD      DE,dest_fcb
 012D CD8801     0092         CALL    make                    ; create the file
                 0093
 0130 119D01     0094         LD      DE,msg_no_room_on_directory
 0133 FEFF       0095         CP      direct_full_flag        ; directory full ?
 0135 2829       0096         JR      Z,exit_program          ; yes, skip to end
                 0097
                 0098 ;       source file open, destination file open
                 0099 ;       copy until end of source
                 0100
 0137 115C00     0101 copy:   LD      DE,source_fcb
 013A CD7C01     0102         CALL    read                    ; read next record
                 0103
```

```
013D FE00          CP      end_of_file_flag           ; end of file ?
013F 200F          JR      NZ,end_of_file             ; skip write if so

             ;     not end of file, write the record

0141 11E301        LD      DE,dest_fcb
```

```
Main program                 RML Z80 Ass V 4.1 J    08-Nov-83    Page 4


0144 CD8201        CALL    write                      ; write record

0147 11B001        LD      DE,msg_no_disc_space
014A FE00          CP      disc_full_flag             ; disc full ?
014C 2012          JR      NZ,exit_program            ; yes, skip to end

014E 18E7          JR      copy                       ; loop until source exhausted



0150         end_of_file:                             ; end of file, close destination

0150 11E301        LD      DE,dest_fcb
0153 CD7001        CALL    close

0156 11C201        LD      DE,msg_write_protected
0159 FEFF          CP      wrt_prtct_flag             ; is disc write protected ?
015B 2803          JR      Z,exit_program             ; shouldn't happen

             ;     copy operation complete, end

015D 11D301        LD      DE,msg_copy_complete

0160         exit_program:                            ; write message given by DE, return
                                                      ; to CCP
0160 0E09          LD      C,fn_print_buffer
0162 CD0500        CALL    bdos

0165 ED7BE101      JP      boot
0169 C9


             ;     End of main program

```

```
File handling routines            RML Z80 Ass V 4.1 J    08-Nov-83     Page 5

                  0143 *H File handling routines
                  0144
  016A 0E0F       0145 open:    LD      C,fn_open_file
  016C CD0500     0146          CALL    bdos
  016F C9         0147          RET
                  0148
  0170 0E10       0149 close:   LD      C,fn_close_file
  0172 CD0500     0150          CALL    bdos
  0175 C9         0151          RET
                  0152
  0176 0E13       0153 delete:  LD      C,fn_delete_file
  0178 CD0500     0154          CALL    bdos
  017B C9         0155          RET
                  0156
  017C 0E14       0157 read:    LD      C,fn_read_file
  017E CD0500     0158          CALL    bdos
  0181 C9         0159          RET
                  0160
  0182 0E15       0161 write:   LD      C,fn_write_file
  0184 CD0500     0162          CALL    bdos
  0187 C9         0163          RET
                  0164
  0188 0E16       0165 make:    LD      C,fn_create_file
  018A CD0500     0166          CALL    bdos
  018D C9         0167          RET
                  0168
```

```
Fixed message area                RML Z80 Ass V 4.1 J    08-Nov-83     Page 6

                  0169 *H Fixed message area
                  0170
  018E 6E6F2073 0171 msg_no_source_file:              defm    'no source file$'
  019D 6E6F2064 0172 msg_no_room_on_directory:        defm    'no directory space$'
  01B0 6F757420 0173 msg_no_disc_space:               defm    'out of data space$'
  01C2 77726974 0174 msg_write_protected:             defm    'write protected?$'
  01D3 636F7079 0175 msg_copy_complete:               defm    'copy complete$'
                0176
```

```
Data area                         RML Z80 Ass V 4.1 J    08-Nov-83     Page 7

                  0177 *H Data area
                  0178
  01E1            0179
  01E3            0180 dest_fcb:        defs    33        ; destination fcb
  0203 =          0181 dest_fcb.current_record        equ     dest_fcb + 32.
                  0182
```

```
Stack space                     RML Z80 Ass V 4.1 J    08-Nov-83    Page 8

                0183 *H Stack space
                0184
 0204           0185           defs    32       ; 16 level stack
                0186
 0224           0187 stack:
                0188
```

```
Symbols                         RML Z80 Ass V 4.1 J    08-Nov-83    Page 9

                0189 *H Symbols
                0190
 0000           0191           end

Symbols:

0005  BDOS               0170  CLOSE              0137  COPY               0176  DELETE
01E3  DEST_FCB           0203  DEST_FCB.CURRENT_  006C  DEST_FILENAME      FFFF  DIRECT_FULL_FLAG
0000  DISC_FULL_FLAG     0150  END_OF_FILE        0000  END_OF_FILE_FLAG   0160  EXIT_PROGRAM
0020  FCB_LENGTH         0080  FILE_BUFFER        FFFF  FILE_NOT_FOUND_FL  0010  FN_CLOSE_FILE
0016  FN_CREATE_FILE     0013  FN_DELETE_FILE     000F  FN_OPEN_FILE       0009  FN_PRINT_BUFFER
0014  FN_READ_FILE       0015  FN_WRITE_FILE      0188  MAKE               01D3  MSG_COPY_COMPLETE
01B0  MSG_NO_DISC_SPACE  019D  MSG_NO_ROOM_ON_DI  018E  MSG_NO_SOURCE_FIL  01C2  MSG_WRITE_PROTECT
01E1  OLD_SP             016A  OPEN               017C  READ               005C  SOURCE_FCB
005C  SOURCE_FILENAME    0224  STACK              0100  TPA                0182  WRITE
FFFF  WRT_PRTCT_FLAG     0000  ZERO

No errors detected
```

APPENDIX B

A SAMPLE FILE DUMP UTILITY

The file dump program on the following pages is slightly more complex than the simple copy program given in Appendix A. The dump program reads an input file, specified in the CCP command line, and displays the contents of each record in hexadecimal format at the console. Note that the dump program saves the CCP's stack upon entry, resets the stack to a local area and restores the CCP's stack before returning directly to the CCP. Thus, the dump program does not perform a warm start at the end of processing.

```
*H DUMP Version 2.0A

; ***********************************************************************
; *                                                                     *
; * DUMP                                                 Version 2.1A   *
  *                                                                     *
; * This program sends a listing of a specified file to the screen in   *
; * hex. It is invoked by a command line of the form :                  *
; *                                                                     *
; *             DUMP X:FILENAME.EXT                                     *
; *                                                                     *
; * which will print the contents of the file on drive X with the       *
; * name and extension. An error message is produced if the file does   *
; * not exist on the specified disc.                                    *
; *                                                                     *
; * Modified:                                                           *
; *     13 Dec 83      to produce COM file directly with ZASM 4.1J      *
; *                                                                     *
; * Adapted from an original version by Digital Research.               *
; *                                                                     *
; * Assembled using ZASM Version 4.1 J                                  *
; ***********************************************************************

```

```
                 0024 *H Equates
                 0025
 0005 =          0026 bdos             equ     0005h    ;bdos entry point
 0100 =          0027 tpa              equ     0100H    ;start of TPA
 005C =          0028 fcb              equ     005CH    ;file control block address
 0080 =          0029 buffer           equ     0080H    ;input disc buffer address
                 0030
                 0031 ;       bdos function numbers
                 0032
 0001 =          0033 fn_console_in    equ     1.
 0002 =          0034 fn_console_out   equ     2.
 0009 =          0035 fn_print_buffer  equ     9.
 000B =          0036 fn_test_kbd      equ     11.
 000F =          0037 fn_open_file     equ     15.
 0010 =          0038 fn_close_file    equ     16.
 0014 =          0039 fn_read_file     equ     20.
                 0040
                 0041 ;       non graphic characters
                 0042
 000D =          0043 cr               equ     0dh      ;carriage return
 000A =          0044 lf               equ     0ah      ;line feed
                 0045
                 0046 ;         file control block definitions
                 0047
 005C =          0048 fcb.dr           equ     fcb+0    ;disc name
 005D =          0049 fcb.fn           equ     fcb+1.   ;file name
 0065 =          0050 fcb.ex           equ     fcb+9.   ;disc file type ( 3 characters )
 0068 =          0051 fcb.rl           equ     fcb+12.  ;file's current reel number
 006B =          0052 fcb.rc           equ     fcb+15.  ;file's record count 0 to 128
 007C =          0053 fcb.cr           equ     fcb+32.  ;current (next) record number 0 to 128
 0020 =          0054 fcb_length       equ     32.
                 0055
                 0056 ;       general equates
                 0057
 FFFF =          0058 file_not_found_flag      equ     -1
 0080 =          0059 buffer_length            equ     128.
 0000 =          0060 zero                     equ     0
 000F =          0061 low_nibble_mask          equ     00001111B
 FFFF =          0062 kbd_ready                equ     -1
 0020 =          0063 space                    equ     ' '
 0000 =          0064 buffer_filled_flag       equ     0
                 0065
```

```
               *H Main program

                       COM
0100                   ORG     tpa

               ;       set up stack

0100 ED73F201          LD      ( old_sp ),SP
0104 313402            LD      SP,stack_top

                       read and print successive buffers

0107 CDC501            CALL    set_up_input_file
010A FEFF              CP      file_not_found_flag
010C 2008              JR      NZ,open_ok

               ;       file not there, give error message and return

010E 11D301            LD      DE,msg_file_not_found
0111 CD9201            CALL    error_message
0114 1833              JR      exit_program

0116           open_ok

               ;        open operation o.k. set buffer index to end

0116 3E80              LD      A,buffer_length
0118 32F101            LD      ( buffer_count ),A

               ;       HL contains number of next byte to print

011B 210000            LD      HL,zero

011E           print_byte_loop:

011E E5                PUSH    HL                              ; save byte count
011F CD9801            CALL    get_next_byte
0122 E1                POP     HL                              ; restore byte count
0123 3824              JR      C,exit_program                  ; carry set by get_next_byte
                                                               ; if end of file

               ;       file not yet empty, print byte in hex

0125 47                LD      B,A                             ; save byte
0126 7D                LD      A,L
0127 E60F              AND     low_nibble_mask                 ; 16 bytes on current line ?
0129 2012              JR      NZ,no_byte_number               ; no, skip byte number

               ;       here if 16 bytes printed on current line. start new line
```

```
                 0115
012B CD6A01      0116           CALL    crlf
012E CD5101      0117           CALL    test_for_key_struck
0131 FEFF        0118           CP      kbd_ready                      ; key struck ?
0133 2814        0119           JR      Z,exit_program                 ; yes, finish dumping
                 0120
```

```
                 0121 ;         no key hit, print byte number
                 0122
0135 7C          0123           LD      A,H
0136 CD8501      0124           CALL    print_hex_number
0139 7D          0125           LD      A,L
013A CD8501      0126           CALL    print_hex_number
                 0127
013D             0128 no_byte_number:
                 0129
013D 23          0130           INC     HL                             ; to next byte number
013E 3E20        0131           LD      A,space
0140 CD5D01      0132           CALL    print_character
0143 78          0133           LD      A,B                            ; recover next byte
0144 CD8501      0134           CALL    print_hex_number
                 0135
0147 18D5        0136           JR      print_byte_loop
                 0137
0149             0138 exit_program:
                 0139
                 0140 ;        end of dump, return to CCP. ( note that jump to 0000 reboots )
                 0141
0149 CD6A01      0142           CALL    crlf
014C ED7BF201    0143           LD      SP,( old_sp )
                 0144
                 0145 ;        stack pointer contains CCP's stack location
                 0146
0150 C9          0147           RET                                    ; to the CCP
                 0148
                 0149 ;        End of main program
                 0150
```

```
                *H Keyboard test routine

 0151           test_for_key_struck:

                ;       Return with -1 in A if key has been struck, otherwise 0
                ;       Does not clear keyboard

 0151 E5                PUSH    HL
 0152 D5                PUSH    DE
 0153 C5                PUSH    BC                          ; environment saved

 0154 0E0B              LD      C,fn_test_kbd
 0156 CD0500            CALL    bdos

 0159 C1                POP     BC
 015A D1                POP     DE
 015B E1                POP     HL                          ; environment restored

 015C C9                RET

```

```
                0171 *H Print routines
                0172
015D            0173 print_character:
                0174
                0175 ;       print a character from register A
                0176
015D E5         0177         PUSH    HL
015E D5         0178         PUSH    DE
015F C5         0179         PUSH    BC
                0180
0160 0E02       0181         LD      C,fn_console_out
0162 5F         0182         LD      E,A
0163 CD0500     0183         CALL    bdos
                0184
0166 C1         0185         POP     BC
0167 D1         0186         POP     DE
0168 E1         0187         POP     HL
                0188
0169 C9         0189         RET
                0190
                0191 ;       -----------------------------
                0192
                0193
016A            0194 crlf:
                0195
                0196 ;       Print carriage return and line feed
                0197
016A 3E0D       0198         LD      A,cr
016C CD5D01     0199         CALL    print_character
                0200
016F 3E0A       0201         LD      A,lf
0171 CD5D01     0202         CALL    print_character
                0203
0174 C9         0204         RET
                0205
                0206 ;       ---------------------------------------
                0207
                0208
0175            0209 print_nibble:
                0210
                0211 ;       prints low nibble of A as hex digit
                0212
0175 E60F       0213         AND     low_nibble_mask
0177 FE0A       0214         CP      10.                         ; is it greater than 9 ?
0179 3004       0215         JR      NC,greater_than_9           ; yes, change to ASCII letter
                0216
                0217 ;       less than or equal to 9, change to ASCII digit
                0218
017B C630       0219         ADD     '0'
```

```
017D 1802      0220         JR      print
               0221
               0222 ;       here if greater than 9
               0223
017F           0224 greater_than_9:
               0225
```

```
Print routines                RML Z80 Ass V 4.1 J    08-Nov-83    Page 7


017F C637      0226         ADD     'A' - 10
               0227
0181 CD5D01    0228 print:  CALL    print_character
               0229
0184 C9        0230         RET
               0231
               0232 ;       ------------------------------------------
               0233
               0234
0185           0235 print_hex_number:
               0236
               0237 ;       print hex number from number in A
               0238
0185 F5        0239         PUSH    AF
0186 0F        0240         RRCA
0187 0F        0241         RRCA
0188 0F        0242         RRCA
0189 0F        0243         RRCA
018A CD7501    0244         CALL    print_nibble
               0245
018D F1        0246         POP     AF
018E CD7501    0247         CALL    print_nibble
               0248
0191 C9        0249         RET
               0250
```

```
Error message routine         RML Z80 Ass V 4.1 J    08-Nov-83    Page 8


               0251 *H Error message routine
               0252
0192           0253 error_message:
               0254
               0255 ;       print error message pointed to on entry by DE
               0256 ;       message terminated with $
               0257
0192 0E09      0258         LD      C,fn_print_buffer
0194 CD0500    0259         CALL    bdos
               0260
0197 C9        0261         RET
               0262
               *H File handling routines

0198           get_next_byte:

               ;       get next byte from file

0198 3AF101            LD      A,( buffer_count )
019B FE80              CP      buffer_length                   ; end of buffer ?
019D 2009              JR      NZ,read_buffer_byte             ; no, get byte from buffer

               ;       buffer empty, read another

019F CDB601            CALL    disc_read

01A2 FE00              CP      buffer_filled_flag              ; buffer filled ?
01A4 2802              JR      Z,read_buffer_byte              ; yes, get next byte

               ;       end of data, return with carry set for eof

01A6 37                SCF
01A7 C9                RET

01A8           read_buffer_byte:

               ;       read the byte at ( buffer + buffer_count )
               ;       buffer count is in register A which is set to zero by BDOS
               ;       after a successful file read. Beware of this register dependancy !

01A8 5F                LD      E,A
01A9 1600              LD      D,0                             ; double precision index to DE
01AB 3C                INC     A                               ; index = index + 1
01AC 32F101            LD      ( buffer_count ),A              ; back to memory

               ;       pointer is incremented, save current file address

01AF 218000            LD      HL,buffer
01B2 19                ADD     HL,DE

               ;       absolute character address is in HL

01B3 7E                LD      A,(HL)

               ;       byte is in the accumulator
               ;       must reset carry flag before return to indicate file not empty

01B4 B7                OR      A                               ; reset carry bit
01B5 C9                RET

               ;       ----------------------------------
```

```
                0312
                0313
 01B6           0314 disc_read:
                0315
                0316 ;        read disc file record
                0317
```

```
File handling routines          RML Z80 Ass V 4.1 J    08-Nov-83    Page 10

 01B6 E5        0318         PUSH    HL
 01B7 D5        0319         PUSH    DE
 01B8 C5        0320         PUSH    BC
                0321
 01B9 115C00    0322         LD      DE,fcb
 01BC 0E14      0323         LD      C,fn_read_file
 01BE CD0500    0324         CALL    bdos
                0325
 01C1 C1        0326         POP     BC
 01C2 D1        0327         POP     DE
 01C3 E1        0328         POP     HL
 01C4 C9        0329         RET
                0330
                0331 ;       ------------------------------
                0332
                0333
 01C5           0334 set_up_input_file:
                0335
                0336 ;       open the file for input
                0337
 01C5 3E00      0338         LD      A,zero                          ; zero to accumulator
 01C7 327C00    0339         LD      ( fcb.cr ),A                    ; clear current record
                0340
 01CA 115C00    0341         LD      DE,fcb
 01CD 0E0F      0342         LD      C,fn_open_file
 01CF CD0500    0343         CALL    bdos
                0344
                0345 ;       -1 in accumulator on return if open error
                0346
 01D2 C9        0347         RET
                0348
```

```
Fixed message area              RML Z80 Ass V 4.1 J    08-Nov-83    Page 11

                0349 *H Fixed message area
                0350
 01D3 6E6F2069  0351 msg_file_not_found:     defm          'no input file present on disc$'
                0352
```

```
Variable area                      RML Z80 Ass V 4.1 J    08-Nov-83    Page 12


                 0353 *H Variable area
                 0354
  01F1           0355 buffer_count:             defs    1         ;input buffer pointer
  01F2           0356 old_sp:                   defs    2         ;entry sp value from ccp
                 0357
```

```
Stack area                         RML Z80 Ass V 4.1 J    08-Nov-83    Page 13


                 0358 *H Stack area
                 0359
  01F4           0360                           defs    64        ;reserve 32 level stack
                 0361
  0234           0362 stack_top:
                 0363
```

```
Symbols                            RML Z80 Ass V 4.1 J    08-Nov-83    Page 14


                 0364 *H Symbols
                 0365
  0000           0366         end

Symbols:

0005  BDOS                0080  BUFFER              01F1  BUFFER_COUNT        0000  BUFFER_FILLED_FLA
0080  BUFFER_LENGTH       000D  CR                  016A  CRLF                01B6  DISC_READ
0192  ERROR_MESSAGE       0149  EXIT_PROGRAM        005C  FCB                 007C  FCB.CR
005C  FCB.DR              0065  FCB.EX              005D  FCB.FN              006B  FCB.RC
0068  FCB.RL              0020  FCB_LENGTH          FFFF  FILE_NOT_FOUND_FL   0010  FN_CLOSE_FILE
0001  FN_CONSOLE_IN       0002  FN_CONSOLE_OUT      000F  FN_OPEN_FILE        0009  FN_PRINT_BUFFER
0014  FN_READ_FILE        000B  FN_TEST_KBD         0198  GET_NEXT_BYTE       017F  GREATER_THAN_9
FFFF  KBD_READY           000A  LF                  000F  LOW_NIBBLE_MASK     01D3  MSG_FILE_NOT_FOUN
013D  NO_BYTE_NUMBER      01F2  OLD_SP              0116  OPEN_OK             0181  PRINT
011E  PRINT_BYTE_LOOP     015D  PRINT_CHARACTER     0185  PRINT_HEX_NUMBER    0175  PRINT_NIBBLE
01A8  READ_BUFFER_BYTE    01C5  SET_UP_INPUT_FILE   0020  SPACE               0234  STACK_TOP
0151  TEST_FOR_KEY_STRU   0100  TPA                 0000  ZERO

No errors detected
```

APPENDIX C

A SAMPLE RANDOM ACCESS PROGRAM

This Appendix contains an extensive example of random access operation. The program listed on the following pages performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled and placed into a file labeled RANDOM.COM, the CCP level command

```
RANDOM X.DAT
```

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found the file is created before the prompt is given. 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  Q
```

where n in an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

```
type data:.
```

The operator then responds 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 the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed and the program returns to the CCP. 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 default file control block at 005CH and the default buffer at 0080H are used in all disc operations. The utility subroutines then follow; these 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.

Again, major improvements could be made to this particular program to enhance its operation. In fact, with some work, this program could evolve into a simple data base management system. One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed 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 "LAST-NAME" field from each record, starting in 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, which is an alphabetical list of LASTNAME fields with their corresponding record numbers (this list is called an inverted index in information retrieval parlance).

If you were to rename the program shown above as QUERY and alter 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 that is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, one can find a particular entry rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, one examines the entry halfway in between and, if not matched, splits either the upper half or the lower half for the next search. You will quickly reach the item you are looking for and find the corresponding record number. You should fetch and display this record at the console, just as was done in the program shown above.

With some more work, you can allow a fixed grouping size that differs from the 128-byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, one randomly accesses the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

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

```
*H RANDOM Version 2.0A

; ***********************************************************************
; *                                                                     *
; * RANDOM                                                Version 2.1A  *
; *                                                                     *
; * This program allows the user to store strings of characters and    *
; * read them back by specifying a random record number for the        *
; * operation. It is designed as a demonstration of random access files *
; * under CP/M.                                                         *
; * The program requires CP/M version 2.0 or better to allow random     *
; * access, but this is tested for automatically.                       *
; * RANDOM is invoked by a command of the form :                        *
; *                                                                     *
; *             RANDOM X:FILENAME.EXT                                   *
; *                                                                     *
; * where the specified file is used for random access. The file will   *
; * be created if it does not already exist. If it does already exist,  *
; * it should have been created by RANDOM for correct functioning of    *
; * the program.                                                        *
; *                                                                     *
; * Modified:                                                           *
; *     13 Dec 83    to produce COM file directly with ZASM 4.1 J       *
; *                                                                     *
; * Developed from a program originally supplied by Digital Research.   *
; *                                                                     *
; * Assembled using ZASM Version 4.1 J                                  *
; ***********************************************************************

```

```
                *H Equates

0005 =          bdos               equ      0005H    ; BDOS entry point
0100 =          tpa                equ      0100H    ; start of TPA
005C =          default_fcb        equ      005CH    ; filename stored here by CCP before entry
0080 =          file_buffer        equ      0080H    ; default file buffer for BDOS

007D =          random_record_number             equ      default_fcb + 33.
007F =          random_record_overflow_byte      equ      default_fcb + 35

                ;        BDOS function numbers

0001 =          fn_console_input         equ     1.
0002 =          fn_console_output        equ     2.
0009 =          fn_print_buffer          equ     9.
000A =          fn_read_buffer           equ     10.
000C =          fn_version               equ     12.
000F =          fn_open_file             equ     15.
0010 =          fn_close_file            equ     16.
0016 =          fn_create_file           equ     22.
0021 =          fn_read_random           equ     33.
0022 =          fn_write_random          equ     34.

                ;        General equates

000D =          cr                       equ     13.              ; carriage return
000A =          lf                       equ     10.              ; line feed
0020 =          version_2.0              equ     20H
FFFF =          file_not_found_flag      equ     -1
FFFF =          no_direct_room_flag      equ     -1
0052 =          read_command             equ     'R'
0057 =          write_command            equ     'W'
0051 =          quit_command             equ     'Q'
007F =          max_chars_to_read        equ     127.
007F =          max_chars_to_print       equ     127.
0000 =          record_terminator        equ     0
0000 =          buffer_terminator        equ     0
0000 =          write_ok_flag            equ     0
0000 =          read_ok_flag             equ     0
0020 =          space                    equ     ' '
0000 =          zero                     equ     0
0005 =          lower_case_bit           equ     5.
0020 =          console_buffer_length    equ     32.
```

```
                0074 *H Main program
                0075
                0076         COM
0100            0077         ORG     tpa
                0078
                0079 ;       set up stack and check CP/M version
                0080
0100 ED73E702   0081         LD      ( old_sp ),SP
0104 310903     0082         LD      SP,stack
                0083
0107 0E0C       0084         LD      C,fn_version
0109 CD0500     0085         CALL    bdos
                0086
010C FE20       0087         CP      version_2.0             ; version 2.0 or better ?
010E 3008       0088         JR      NC,version_ok           ; yes, carry on
                0089
                0090 ;       bad version, message and go back
                0091
0110 112302     0092         LD      DE,msg_bad_version
0113 CDE301     0093         CALL    print_message
                0094
0116 1851       0095         JR      exit_program
                0096
0118            0097 version_ok:
                0098
                0099 ;       correct version for random access
                0100
0118 0E0F       0101         LD      C,fn_open_file
011A 115C00     0102         LD      DE,default_fcb
011D CD0500     0103         CALL    bdos
                0104
0120 FEFF       0105         CP      file_not_found_flag             ; file found ?
0122 2014       0106         JR      NZ,file_ready                   ; yes, carry on
                0107
                0108 ;       cannot open file so create it
                0109
0124 0E16       0110         LD      C,fn_create_file
0126 115C00     0111         LD      DE,default_fcb
0129 CD0500     0112         CALL    bdos
                0113
012C FEFF       0114         CP      no_direct_room_flag             ; directory full ?
012E 2008       0115         JR      NZ,file_ready                   ; no, carry on
                0116
                0117 ;       cannot create file, directory full
                0118
0130 114E02     0119         LD      DE,msg_directory_full
0133 CDE301     0120         CALL    print_message
                0121
0136 1831       0122         JR      exit_program                    ; back to CCP
```

```
0138        file_ready:
0138        get_next_command:

            ;       file is ready for processing

```

```
0138 CDEE01    0129         CALL    get_command
               0130
               0131 ;       set random record number. next record number is in HL
               0132 ;       on return from read_command
               0133
013B 227D00    0134         LD      ( random_record_number ),HL
013E 217F00    0135         LD      HL,random_record_overflow_byte
0141 3600      0136         LD      (HL),0                          ; clear high byte if set
               0137
               0138 ;       command letter may be R(ead, W(rite or Q(uit and is in A
               0139 ;       on return from read_command
               0140
0143 FE51      0141         CP      quit_command                    ; quit ?
0145 281A      0142         JR      Z,quit_processing               ; yes, close file
               0143
               0144 ;       command was not quit.
               0145
0147 FE57      0146         CP      write_command                   ; write record ?
0149 2811      0147         JR      Z,write_random_record           ; yes, deal with it
               0148
               0149 ;       command was not write.
               0150
014B FE52      0151         CP      read_command                    ; read record ?
014D 2808      0152         JR      Z,read_random_record            ; yes, deal with it
               0153
               0154 ;       command is not valid. print error message and loop back
               0155
014F 116E02    0156         LD      DE,msg_invalid_command
0152 CDE301    0157         CALL    print_message
0155 18E1      0158         JR      get_next_command
               0159
0157           0160 read_random_record:
               0161
0157 CD9D01    0162         CALL    read_record
015A 18DC      0163         JR      get_next_command
               0164
015C           0165 write_random_record:
               0166
015C CD6E01    0167         CALL    write_record
015F 18D7      0168         JR      get_next_command
               0169
0161           0170 quit_processing:
               0171
0161 0E10      0172         LD      C,fn_close_file
0163 115C00    0173         LD      DE,default_fcb
0166 CD0500    0174         CALL    bdos
               0175
0169           0176 exit_program:
               0177
```

```
0169 ED7BE702 0178          LD      SP,( old_sp )
016D C9       0179          RET                                     ; to CCP
              0180
              0181 ;        End of main program
              0182
```

```
                  *H Random write routine

016E              write_record:

                  ;       this is a random write. fill the buffer from the keyboard
                  ;       until carriage return encountered, or 127 characters read.

016E 116102               LD      DE,data_prompt
0171 CDE301               CALL    print_message

0174 067F                 LD      B,max_chars_to_read
0176 218000               LD      HL,file_buffer

0179              get_keyboard_character:

                  ;       read next character to buffer

0179 C5                   PUSH    BC                           ; save counter
017A E5                   PUSH    HL                           ; next destination
017B CDCB01               CALL    get_character                ; character to A

017E E1                   POP     HL                           ; restore next to fill
017F C1                   POP     BC                           ; restore counter
0180 FE0D                 CP      cr                           ; end of line ?
0182 2804                 JR      Z,end_of_read_char_loop      ; yes, jump out of loop

                  ;       not end, store character

0184 77                   LD      (HL),A
0185 23                   INC     HL                           ; next to fill
0186 10F1                 DJNZ    get_keyboard_character       ; loop back

0188              end_of_read_char_loop:

0188 3600                 LD      (HL),record_terminator

                  ;       write the record to the next record number

018A 0E22                 LD      C,fn_write_random
018C 115C00               LD      DE,default_fcb
018F CD0500               CALL    bdos

0192 FE00                 CP      write_ok_flag                ; write successful ?
0194 2806                 JR      Z,write_successful           ; yes, return

                  ;       write unsuccessful. print message

0196 119A02               LD      DE,msg_disc_full
0199 CDE301               CALL    print_message
```

```
019C            write_successful:

019C C9                 RET

```

```
                0237 *H Random read routine
                0238
019D            0239 read_record:
                0240
019D 0E21       0241         LD      C,fn_read_random
019F 115C00     0242         LD      DE,default_fcb
01A2 CD0500     0243         CALL    bdos
                0244
01A5 FE00       0245         CP      read_ok_flag                ; read successful ?
01A7 2808       0246         JR      Z,read_successful           ; yes, print buffer
                0247
                0248 ;       read unsuccessful, print message
                0249
01A9 11AF02     0250         LD      DE,msg_no_record_exists
01AC CDE301     0251         CALL    print_message
01AF 1819       0252         JR      end_of_random_read
                0253
01B1            0254 read_successful:
                0255
01B1 CDD801     0256         CALL    crlf                        ; new line
                0257
01B4 067F       0258         LD      B,max_chars_to_print
01B6 218000     0259         LD      HL,file_buffer
                0260
01B9            0261 print_record_loop:
                0262
01B9 7E         0263         LD      A,(HL)                      ; next character
01BA 23         0264         INC     HL                          ; next to get
01BB FE00       0265         CP      record_terminator           ; end of record ?
01BD 280B       0266         JR      Z,end_of_random_read        ; yes, skip to end
                0267
01BF C5         0268         PUSH    BC                          ; save counter
01C0 E5         0269         PUSH    HL                          ; save next to get
01C1 FE20       0270         CP      space                       ; graphic ?
01C3 D4D101     0271         CALL    NC,print_character          ; skip output if not
                0272
01C6 E1         0273         POP     HL
01C7 C1         0274         POP     BC
01C8 10EF       0275         DJNZ    print_record_loop
                0276
01CA            0277 end_of_random_read:
                0278
01CA C9         0279         RET
                0280
                *H Console input and output

01CB            get_character:

                ;       read next console character into A

01CB 0E01               LD      C,fn_console_input
01CD CD0500             CALL    bdos

01D0 C9                 RET

                ;       -------------------------------------


01D1            print_character:

                ;       write character from A to console

01D1 0E02               LD      C,fn_console_output
01D3 5F                 LD      E,A
01D4 CD0500             CALL    bdos

01D7 C9                 RET

                ;       --------------------------------------


01D8            crlf:

                ;       send crlf to console

01D8 3E0D               LD      A,cr
01DA CDD101             CALL    print_character

01DD 3E0A               LD      A,lf
01DF CDD101             CALL    print_character

01E2 C9                 RET

                ;       ----------------------------------------


01E3            print_message:

                ;       print the buffer addressed by DE until $

01E3 D5                 PUSH    DE
01E4 CDD801             CALL    crlf
01E7 D1                 POP     DE
```

```

01E8 0E09          LD      C,fn_print_buffer
01EA CD0500        CALL    bdos

01ED C9            RET

```

```
                ;       ------------------------------------


01EE            get_command:

                ;       read the next command from the console to the console buffer

01EE 118A02             LD      DE,command_prompt
01F1 CDE301             CALL    print_message

01F4 0E0A               LD      C,fn_read_buffer
01F6 11C502             LD      DE,console_buffer
01F9 CD0500             CALL    bdos                                ; read command

01FC CD0002             CALL    process_command_line

01FF C9                 RET


                ;       --------------------------------

0200            process_command_line:

                ;       capture record number from command line into HL ignoring
                ;       overflow ( i.e. mod 2 ^ 16 ). if no number is present,
                ;       assume record zero.
                ;       return with command character in A

0200 210000             LD      HL,zero                             ; start with record zero
0203 11C702             LD      DE,command_line                     ; command line

0206            read_command_character:

0206 1A                 LD      A,( DE )                            ; next command character
0207 13                 INC     DE                                  ; to next command position
0208 FE00               CP      buffer_terminator                   ; end of command line ?
020A 2816               JR      Z,end_of_command_line               ; yes, skip to end

020C D630               SUB     '0'
020E FE0A               CP      10.                                 ; is character a digit ?
0210 300C               JR      NC,end_of_digits                    ; no, restore command char.
                                                                    ; and return

                ;       add in next digit

0212 29                 ADD     HL,HL                               ; HL = HL times 2
0213 4D                 LD      C,L
0214 44                 LD      B,H                                 ; BC = HL times 2
0215 29                 ADD     HL,HL                               ; HL = HL times 4
```

```
0216 29          0385            ADD      HL,HL                              ; HL = HL times 8
0217 09          0386            ADD      HL,BC                              ; HL = HL times 8 + HL times 2
                 0387                                                        ;    = HL times 10
0218 4F          0388            LD       C,A
0219 0600        0389            LD       B,zero                             ; BC = next digit
021B 09          0390            ADD      HL,BC
```

```
                 0391
021C 18E8        0392            JR       read_command_character
                 0393
021E             0394 end_of_digits:
                 0395
021E C630        0396            ADD      '0'                                ; restore command
0220 CBAF        0397            RES      lower_case_bit,A                   ; ensure upper case
                 0398
0222             0399 end_of_command_line:
                 0400
0222 C9          0401            RET
                 0402
```

```
                 0403 *H Fixed message area
                 0404
0223 736F7272 0405 msg_bad_version:          defm    'sorry, you need cp/m version 2.2 or better$'
024E 6E6F2064 0406 msg_directory_full:       defm    'no directory space$'
0261 74797065 0407 data_prompt:              defm    'type data > $'
026E 696E7661 0408 msg_invalid_command:      defm    'invalid command, try again.$'
028A 6E657874 0409 command_prompt:           defm    'next command > $'
029A 6E6F206D 0410 msg_disc_full:            defm    'no more room on disc$'
02AF 7265636F 0411 msg_no_record_exists:     defm    'record does not exist$'
              0412
```

```
                 0413 *H Data area
                 0414
02C5 20          0415 console_buffer:          defb    console_buffer_length
02C6             0416 console_buffer_size:     defs    1
02C7             0417 command_line:            defs    32
                 0418
02E7             0419 old_sp:                  defs    2
02E9             0420                          defs    32          ;16 level stack
                 0421
0309             0422 stack:
                 0423
                 0424
```

```
                0425 *H Symbols
                0426
  0000          0427            end

Symbols:

0005  BDOS                 0000  BUFFER_TERMINATOR    02C7  COMMAND_LINE         028A  COMMAND_PROMPT
02C5  CONSOLE_BUFFER       0020  CONSOLE_BUFFER_LE    02C6  CONSOLE_BUFFER_SI    000D  CR
01D8  CRLF                 0261  DATA_PROMPT          005C  DEFAULT_FCB          0222  END_OF_COMMAND_LI
021E  END_OF_DIGITS        01CA  END_OF_RANDOM_REA    0188  END_OF_READ_CHAR_    0169  EXIT_PROGRAM
0080  FILE_BUFFER          FFFF  FILE_NOT_FOUND_FL    0138  FILE_READY           0010  FN_CLOSE_FILE
0001  FN_CONSOLE_INPUT     0002  FN_CONSOLE_OUTPUT    0016  FN_CREATE_FILE       000F  FN_OPEN_FILE
0009  FN_PRINT_BUFFER      000A  FN_READ_BUFFER       0021  FN_READ_RANDOM       000C  FN_VERSION
0022  FN_WRITE_RANDOM      01CB  GET_CHARACTER        01EE  GET_COMMAND          0179  GET_KEYBOARD_CHAR
0138  GET_NEXT_COMMAND     000A  LF                   0005  LOWER_CASE_BIT       007F  MAX_CHARS_TO_PRIN
007F  MAX_CHARS_TO_READ    0223  MSG_BAD_VERSION      024E  MSG_DIRECTORY_FUL    029A  MSG_DISC_FULL
026E  MSG_INVALID_COMMA    02AF  MSG_NO_RECORD_EXI    FFFF  NO_DIRECT_ROOM_FL    02E7  OLD_SP
01D1  PRINT_CHARACTER      01E3  PRINT_MESSAGE        01B9  PRINT_RECORD_LOOP    0200  PROCESS_COMMAND_L
0051  QUIT_COMMAND         0161  QUIT_PROCESSING      007D  RANDOM_RECORD_NUM    007F  RANDOM_RECORD_OVE
0052  READ_COMMAND         0206  READ_COMMAND_CHAR    0000  READ_OK_FLAG         0157  READ_RANDOM_RECOR
019D  READ_RECORD          01B1  READ_SUCCESSFUL      0000  RECORD_TERMINATOR    0020  SPACE
0309  STACK                0100  TPA                  0020  VERSION_2.0          0118  VERSION_OK
0057  WRITE_COMMAND        0000  WRITE_OK_FLAG        015C  WRITE_RANDOM_RECO    016E  WRITE_RECORD
019C  WRITE_SUCCESSFUL     0000  ZERO

No errors detected
```

APPENDIX D

USE OF MEMORY BY CP/M

Area	Memory Locations	(Hex)	Contents
System Area	0-2	RST 0	Jump to BIOS warm start entry point (shared with COS and ROS)
	3		Reserved
	4		System use (associated with logged-in drive)
	5-7		Standard BDOS function entry point (shared with COS and ROS)
	8	RST 8	Not used - reserved
	E-F		Top of physical RAM + 1
	10-17	RST 10	Used at power-up
	18-1F	RST 18	Reserved
	20-27	RST 20	Used by COS/ROS
	28-2F	RST 28	COS/ROS CALR mechanism
	30-37	RST 30	COS/ROS EMT mechanism
	38-3F	RST 38	Break to Front Panel - also used by DDT and ZSID
	40-43		Disc map (system use only)
	44-4F		Reserved (system use only)
	50-5B		Not used (disc maps)
	5C-7C		File Control Block (FCB) area (default)
	66-69		Front Panel single step - locations restored after single step - cannot single step through file access
	7D-7F		Random record position (default)
	80-FF		DMA buffer area (128 bytes) for input and output (default)
Transient Program Area	100...		Area where programs are loaded

Advanced use of CP/M 5.1
Appending to file 2.12
Assembling your programs 1.5

Basic Disc Operating System 1.2
Basic Input/Output System 1.2
BDOS 1.2
BDOS functions 2.1
- access drive 4.14
- close file 2.23
- compute file size 2.32
- configuration tables 4.20
- console input 2.5, 2.17
- console output 2.4, 2.18
- delete file 2.25
- direct console I/O 2.6, 2.19
- free drive 4.15
- get ADDR(Alloc) 2.28
- get ADDR(disc parms) 2.29
- get allocation vector add. 4.8
- get console status 2.6, 2.21
- get disc parameter address 4.8
- get network status 4.20
- get read/only vector 2.28
- introduction 4.1
- list output 2.6, 2.19
- lock record 4.15
- log off 4.19
- make file 2.26, 4.5
- open file 2.23, 4.5
- print string 2.19
- punch output 2.6, 2.18
- read console buffer 2.5, 2.20
- read random 2.30
- read sequential 2.25
- reader input 2.6, 2.18
- rename files 2.26
- reset disc system 2.22
- reset drive 2.33
- return current disc 2.27
- return log-in vector 2.27
- return version number 2.22, 4.4
- search for first 2.24
- search for next 2.24
- select disc 2.22
- set BDOS error mode 4.18
- set default password 4.21
- set file attributes 2.29
- set random record 2.32
- set/get user code 2.29
- simple devices 2.3
- system reset 2.7, 2.17
- unlock record 4.17
- write protect disc 2.28
- write random 2.31
- write random (zero fill) 2.33
- write sequential 2.25
Binary files 2.8
BIOS 1.2
BIOS entry point subroutines 5.4

CCP 1.2, 4.3
Closing files 2.7, 2.12
Command line 1.2
Command line tail 2.13
Configuration tables 4.9, 4.5
CP/M
- interface with COS/ROS 1.1
- interfaces 1.1
- structure 1.1
- system entry points 2.1, 1.1
- use of memory A.1
CP/M 2 system interface
- introduction 2.1
- operating system functions 2.17
- transporting software 2.16
- using BDOS disc functions 2.7
- using simple BDOS functions 2.3
CP/M interfaces
- user interface to CCP 1.2
CP/NET 4.1
CP/NET BDOS functions
- access drive 4.14
- configuration tables 4.20
- free drive 4.15
- get network status 4.20
- introduction 4.1
- lock record 4.15
- log off 4.19
- set BDOS error mode 4.18
- set default password 4.21
- unlock record 4.17
CP/NET stations 4.1
CP/NOS 4.2
Current user number 2.29
Currently-selected drive 2.15

DDT
- commands 3.4
- initiating 3.2
Debugging your programs 3.1
- Front Panel 1.7
Default DMA buffer 2.13
Default error mode(CP/NET) 4.8
Delete file 2.14

Device handlers
 - adding 5.1
 - connecting to CP/M 5.1
Device mapping across network 4.9
Direct Memory Access address 2.9
Directory 2.7
Directory code 2.23
Directory operations 2.14
Disc data buffer 2.9
Disc parameter block 2.29
Disc parameter block (DPB) 5.7
Disc parameter header 5.6
Disc parameter tables 5.6
Disc protection 2.14
DMA address 2.9
DPB 5.7
Drive select code 2.8
Duplicate filenames 2.26

EMT functions 1.2
EMT instructions 2.2
Emulator traps 1.2
Error handling 2.2
Exiting from your program 2.7

Fault-tolerant software 2.2
FCB 1.2, 2.9
FDOS 1.2
File
 - closing 2.7, 2.8
 - creation 2.7
 - end of file 2.8
 - random read from 2.8
 - random write to 2.8
 - sequential read 2.8
 - sequential write 2.7
 - updating 2.8
File attributes 2.29
File control block (FCB) 2.9
File control block(FCB) 1.2
File handling under CP/NET 4.4
File locking 4.5
File protection 2.14
File type 2.8
Filename 2.8
Files 2.7
Front Panel 1.7
Function
 - 0 2.17
 - 1 2.17
 - 2 2.18
 - 3 2.18
 - 4 2.18
 - 5 2.19
 - 6 2.19
 - 9 2.19
 - 10 2.20
 - 11 2.21
 - 12 2.22
 - 13 2.22
 - 14 2.22
 - 15 2.23
 - 16 2.23
 - 17 2.24
 - 18 2.24
 - 19 2.25
 - 20 2.25
 - 21 2.25
 - 22 2.26
 - 23 2.26
 - 24 2.27
 - 25 2.27
 - 26 2.27
 - 27 2.28
 - 28 2.28
 - 29 2.28
 - 30 2.29
 - 31 2.29
 - 32 2.29
 - 33 2.30
 - 34 2.31
 - 35 2.32
 - 36 2.32
 - 37 2.33
 - 38 4.14
 - 39 4.15
 - 40 2.33
 - 42 4.15
 - 43 4.17
 - 45 4.18
 - 65 4.19
 - 68 4.20
 - 69 4.20
 - 71 4.20
 - 106 4.21
Function code 2.2

HEX files
 - loading 1.6
High-level languages
 - use with CP/M 2.2
HIMEM 1.4

Jump vector 5.1

List device buffer 4.10
Loading a HEX file 1.6

Loading and running your
 - programs 1.6
Loading your programs 1.6
Logged-in drive 2.15
Logical extents 2.8

Make file 2.9
Memory available to programs 1.4
Memory used by COS/ROS 1.5
Microsoft M80 assembler 1.6
MP/M 4.1

NDOS 4.3
Network Disc Operating System 4.3

Opening files 2.7, 2.12

Page zero
 - reserved locations 5.10
Passing filenames to programs 2.13
Password protection (CP/NET) 4.11
Patching programs
 - using DDT 3.2
 - using Front Panel 3.1
Portable software 2.16
Producing program source 1.5
Program development 1.4

Random file operations 2.12
Random record field 2.30
Random record number 2.10
Records 2.7
Reducing size of TPA 5.1
Rename file 2.14
Reserved locations in
 - page zero 5.10
Reset disc system 2.16
Reset drive 2.16
Return and display error
 - mode (CP/NET) 4.8
Return error mode(CP/NET) 4.8
Running your programs 1.6

Sequential read from file 2.9
Sequential write to file 2.9
Server lock list 4.15
Server network status byte 4.11
Size of file 2.16
SNIOS 4.3
Software interrupts 1.2
Source files 2.8
Source program 1.6
Space available for programs 1.4
Start address of programs 1.4
Station configuration table 4.10
Station Network I/O System 4.3
Station processor ID 4.10
Station status byte 4.10
System attributes 2.29
System entry points 1.1
System lock list 4.14
System parameter area 1.2

Temporary filenames 4.12
Text editor(TXED) 1.5
Transient command 1.2
Transient program area 1.2
TXED 1.5

Use of memory by CP/M A.1
User interface to CCP 1.2

Wildcard facility 2.14

ZASM assembler 1.5

USER'S COMMENTS

To help Research Machines to produce the highest quality microcomputers, supporting software, and technical publications, we like to hear from users about their experiences with our products.

Do share your thoughts with us by jotting them down on the tear-off form on the next page. You can leave out your personal details, if you want to. Fold the form in two, seal it with a piece of adhesive tape, and put it in the post.

If you would like to give more information than we have allowed room for on the form, we will be very pleased to receive a separate letter from you. You can even use the form to ask for a post-paid envelope, if you wish.

Additional information will be most useful, if you give as much detail as possible about your hardware configuration, software version number, or manual title, so that we can relate your comments to the correct products.

Seal with self-adhesive tape (not staples) along this edge.

RESEARCH MACHINES
MICROCOMPUTER SYSTEMS

Fold along this line.

Postage
will be
paid by
licensee

Do not affix Postage Stamps if posted in
Gt Britain, Channel Islands, N Ireland
or the Isle of Man

1

BUSINESS REPLY SERVICE
Licence No OF32.

TECHNICAL PUBLICATIONS DEPT
RESEARCH MACHINES LTD
PO BOX 75 OXFORD
OX2 0BR

USER'S COMMENTS **CP/M & CP/NET PROGRAMMERS GUIDE PN 12084**

User's comments help us to improve our products. If you would like to make any comments, please use this reply-paid form.

Your comments:

Research Machines may use this information in any way believed to be appropriate and without obligation.

Although it is not essential, it would be helpful if you gave the following information:

Name..

Organization..

Address...

.... Post Code......................

System: 380Z / 480Z / Network ☐ **Cassette / 5.25" discs / 8" discs**
(Delete as necessary)