ZCPR3

THE MANUAL

RICHARD CONN

NEW YORK ZOETROPE

ZCPR3 The Manual

Library of Congress Cataloging in Publication Number:
84-061790

ISBN 0-918432-59-6

New York Zoetrope
80 East 11th Street
New York, NY 10003

ZCPR3 The Manual was set in H-P Times Roman. The book was printed and bound by Maple-Vail Book Manufacturing Group at their York, PA. plant.

Production: Mitch Knauffs, Chris Terry, Richard White.

Printed in the United States of America
First Printing: June, 1985
5 4 3 2 1

FOREWORD

This book has been greatly needed since the release of ZCPR3 about six months ago. It took about eight months to write ZCPR3 and its online documentation. Surprisingly, it took six months to produce this book; to say the least, this book took much more effort and time than anticipated. But I feel it was worth it, and I thank the community of users who have patiently waited for the book to be completed.

I thank Chris Terry for his monumental effort in performing the technical editing for this book. I also thank Frank Gaude of Echelon for his support. Finally, I thank my parents and my second family for the support and encouragement they gave me throughout the trial of creating the book. I believe that the end result is worth it.

DEDICATED
to my parents and my second family

Richard Conn
December 17, 1984

TABLE OF CONTENTS

Section 1

Using ZCPR3 and Command Definitions

"As long as there were no machines, programming was not a problem at all; when we had a few weak computers, programming became a mild problem and now that we have gigantic computers, programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them—it has created the problem of using its product."
E.W. Dijkstra, ACM Turing Award Lecture, 1972

"In the development of our understanding of complex phenomena, the most powerful tool available to the human intellect is abstraction. Abstraction arises from a recognition of similarities between certain objects, situations, or processes in the real world, and the decision to concentrate on these similarities, and to ignore for the time being the differences."
C.A.R. Hoare, Notes on Data Structuring.

This book is divided into three sections. Section 1 deals with learning and using the system. In Chapters 1-2, the philosophy of ZCPR3 is presented, ZCPR3 and CP/M 2.2 are compared, and the basic concepts of ZCPR3 are outlined. Chapter 3 presents the ZCPR3 Toolset in detail; all of the commands are listed in alphabetic order, with their syntax, their application, and examples of how to use them. After reading these chapters, a person should be ready to use the ZCPR3 System. Chapters 4 through 8 discuss the online Help, Menu, and Shell subsystems, together with two of the major tools (VFILER and DU3) in somewhat more detail than could be given in Chapter 3.

Section 2 discusses the internal structure and workings of some parts of ZCPR3; this section, comprising Chapters 9 through 15, is aimed at the systems programmer who will be using, installing, and adapting to his own needs a ZCPR system.

Section 3 contains detailed instructions for installing a ZCPR3 system. Chapter 16 presents an overview of the installation process, which is complex and requires familiarity with assembly language programming and the interface requirements of CP/M. Chapter 17 offers guidance on how to select the many features available if you don't have enough memory or disk space to accommodate all of them. Chapters 18 through 22 describe in detail the installation procedures and the ZCPR3 tools that make life easier for the installer.

Do not be put off by the complexities of Section 3. If you merely want to use ZCPR3 with all its flexibility and added power, you can obtain a self-installing version, called Z3-DOT-COM from Echelon, Inc. This version pokes and pries into your own system until it has all the required information, and then begins the installation process automatically, reporting on your console as it completes installation of each feature and utility. Two very simple commands entered from the keyboard do all the work for you.

But if you are installing ZCPR3 from the source files, or if you want to adapt the system to meet special needs of your own, then you will need to study Section 3 very closely.

Throughout this book a working knowledge of CP/M 2.2 is assumed. Further, since ZCPR3 is *not* compatible with CP/M versions earlier than 2.2, the reader should assume that all references to "CP/M" mean "CP/M version 2.2."

1 ZCPR3 AND CP/M

This chapter explains the ideas and motivation behind ZCPR3. It describes, in some detail, what ZCPR3 offers to the user and what the user needs to understand in order to make effective use of ZCPR3.

A First Look at ZCPR3

To see how ZCPR3 meshes with CP/M, let us first take a look at the memory map of figure 1-1, which shows the components of a standard CP/M system alongside those of a ZCPR3-based system.

```
Address              CP/M System                ZCPR3 System


                                             ---------------------
                                             | Extended Features |
High Memory -> ---------------------         ---------------------
               | BIOS              |         | Modified BIOS     |
BDOS+0E00H  -> ---------------------         ---------------------
               | CP/M 2.2 BDOS     |         | CP/M 2.2 BDOS     |
CCP +0800H  -> --------------------- --      ---------------------
               | CP/M 2.2 CCP      | T       | ZCPR3             |
CCP Base    -> --------------------- P       ---------------------
               | Scratch Area      | A*      | Scratch Area      |
100H        -> --------------------- --      ---------------------
               | CP/M Buffers et al|         | ZCPR3 Buffers et al|
  0H        -> ---------------------         ---------------------
```

*TPA = Transient Program Area, which covers the Scratch Area and the CCP or ZCPR3

Figure 1-1. Components of standard CP/M and ZCPR3 systems

A ZCPR3-based system is structured in almost the same way as a CP/M-based system. For all intents and purposes, ZCPR3 looks like CP/M to a program designed to run under CP/M. The differences are in the way ZCPR3 looks to a program designed to run under ZCPR3 and the way ZCPR3 looks to the user. A program designed to run under ZCPR3 can extract much more information about the user's system from ZCPR3 than it could from CP/M, and it can automatically configure itself to use the resources of the user's system in ways which it could not do under CP/M.

CP/M As an Operating System

CP/M is an *Operating System*: that is, a computer program whose function is to manage the resources of the computer and to provide services in response to standardized requests from *application programs* (which manipulate data in ways that serve the user's specific needs). All computers have four basic resources to be managed by the operating system:

o Memory
o Processors and Processes (a Process is a running Program)
o Devices
o Information

Memory Management. CP/M does very little in the way of *Memory Management.* It merely defines the basic memory structure as shown in figure 1-1; it does not allocate sections of the memory to application programs or do any of the partitioning usually associated with the memory management function.

Process Management. CP/M does no *Process Management.* There is only one processor, so no processor management is needed. Only one process at a time can run, so CP/M simply starts the process and then relinquishes all control. The process then has complete control over the entire microcomputer, and CP/M does nothing to stop it from doing anything it wishes to do. Co-routines or time-sharing between users on the basis of a clock interrupt are sometimes found, but in such cases the allocation of time slices to users is a function of a separate process control program, not of CP/M.

Device Management. The beauty of CP/M is the way it does device and information management. *Device Management*is performed by the *Basic Input-Output System* (BIOS) of CP/M. The BIOS provides a standardized, hardware-independent interface to the devices attached to the microcomputer. The routines controlling these devices may be accessed by way of a table of jump instructions located at the beginning of the BIOS. The parameters passed to these routines, the parameters returned by these routines, and the functions performed by these routines are precisely defined. The applications programmer does not need to know *how* the routines perform their functions, but only what they do and how to communicate with them. This is a *process box* , or *black box* concept, illustrated in figure 1-2.

```
Input Parameters    |
are precisely known |                 The Process Box is
                    V                 a "Black Box," and
        ---------------------         it is not necessary
        | Function is        |   <--  to know what is in
        | Precisely Known    |        the box in order
        ---------------------         to use it
                    |
  Output Parameters |
are precisely known |
                    V
```

Figure 1-2. Process Box

For example, the fourth entry in the BIOS Jump Table accesses the Console Input Routine. To obtain a character from the user's console, regardless of what type of device the console may be (such as a CRT or printing terminal), an application program has only to make a subroutine call to address BIOS + 9 (each Jump Table Entry is three bytes long); the BIOS will then return the next character from the console in the A register. To output a character to the console, the software need only place that character into the C register and make a call to BIOS + 12. The jump table, which is in a constant location relative to the start of CP/M, ensures that the correct driver routine will be accessed, regardless of where CP/M is located in the memory (minimum memory size is 22K, maximum is 4K).

The BIOS performs all of the functions necessary for CP/M (and the programs which run under CP/M) to control and communicate with the disk subsystem and most commonly used peripherals such as printers, modems, and so on. Surprisingly, only seventeen general-purpose functions are required to provide the hardware interfaces necessary to perform all character and disk I/O under CP/M. They are:

1. Initialization Functions
   o Cold boot initialization (when the system is first turned on)
   o Warm boot initialization (performed periodically after the system is turned on)
2. Character Input/Output Functions
   o Console status (check for availability of a character at the console)
   o Console input
   o Console output
   o List status (check to see if List Device is ready to output the next character)
   o List (printer) output
   o Auxiliary output
   o Auxiliary input
3. Disk Input/Output Functions
   o Home drive (move head to Track 0)
   o Select drive (which drive to use)
   o Select track
   o Select sector
   o Select memory address to read into or write from
   o Read block (at selected track and sector) into memory
     (at selected memory address)
   o Write block (to selected track and sector) from memory
     (at selected memory address)
   o Logical-to-physical sector translation (for efficiency of disk use)

The BIOS, then, creates a *virtual machine* : that is, a hypothetical computer on which all CP/M software runs (including CP/M itself). This hypothetical computer always behaves in the same manner and has the same logical interface to application programs, regardless of the actual hardware used to implement it. It makes no difference whether 5.25" floppy disks storing 100K per disk, 8" floppy disks storing 600K per disk, or 8" Winchester hard disks storing 5M per disk are used. The software talks to all of these devices in the same way, and this makes such software transportable *at the binary level* between any two microcomputers running CP/M. The only proviso is that the application program must use the standard CP/M function calls when requesting service; direct calls to the BIOS via the jump table may result in loss of portability to some systems.

Information Management. Information management, in the CP/M context, consists of the control of files on disk. CP/M shines here too, extending the virtual machine concept to the management of files on disk. The *Basic Disk Operating System* (BDOS) portion of CP/M creates this file-oriented virtual machine. To illustrate this point, some (but by no means *all*) of the functions provided by the BDOS are:

o Reset disk system
o Select disk
o Create file (actually, create a directory entry for a file)
o Open file (make a file ready for reading or writing)
o Close file (terminate the read/write process)
o Delete file
o Rename file
o Set memory address to read into or write from
o Read next block from file
o Write next block into file

Note the similarity between these BDOS functions and the BIOS disk functions. These BDOS functions are accessed in a different way from the BIOS, but the process box concept is maintained. All one needs to know are the input parameters, the output parameters, and what the function performed is. Once more, transportability is realized *at the binary level* , but this time it is with respect to the information manipulated by the computer–a more general concept than merely performing disk and character I/O. It is at this more abstract level that the virtual machine makes possible the exchange and sale of software. In effect, the creation of CP/M spawned an industry based on a feature found only in computer systems running the UNIX operating system: the exchange of software regardless, by and large, of the actual computer hardware involved and independent of any one computer hardware manufacturer. CP/M and UNIX differ in that CP/M restricts the microprocessor used in the computer, while UNIX makes no restriction. It was only this feature, combined with the open architecture and inexpensive hardware of the machines, that allowed so many third-party software vendors to develop programs that fired the imagination and made the industry viable.

Where ZCPR3 Fits In

As we can see from Figure 1-1, the virtual machine of CP/M is left more-or-less intact in the ZCPR3 environment. The BDOS is unchanged, and though the BIOS is modified, the changes are minor and the interfaces are left unaffected. Hence, under ZCPR3, we are dealing with the same virtual machine, so that software which ran under CP/M 2.2 will also run under ZCPR3, except in a few rare cases in which the software calls on the CCP to perform some functions (in which cases ZCPR3 may or may not work).

2 Basic ZCPR3 Concepts

ZCPR3 provides a more convenient and significantly more powerful human interface than the standard CCP (command processor) of CP/M. The facilities provided by ZCPR3 are described below. In this discussion it is assumed that all the available features have been installed; however, the user can choose to install only those features that will be useful to him.

Directories

Like CP/M, ZCPR3 is able to address up to sixteen logical disks, each containing up to thirty-two user areas. A *directory* under ZCPR3 defines a user area on a disk, and is identified by one of two methods. The first method is the combination of the disk letter and the the user area number (e.g., A10 for disk A, user area 10); this will be referred to as the "DU (disk/user) form." The second method is to use a mnemonic (such as JEFF, which could be assigned to disk B, user area 5). The naming of a directory will hereinafter be referred to as the "DIR form." By convention, the name ROOT is assigned to Disk A/User 15.

The directory is a *logical* concept. In a multi-user system it serves to separate the files of the various users; in a single-user system, it is valuable for grouping together the files related to a specific project and separating them from those of other projects. It is important to note that, because ZCPR3 uses the file management facilities of the CP/M BDOS, there is only one *physical* directory on each logical disk. This physical directory contains the entries for all files in all user areas on the disk; a user number is a part of each directory entry and associates the file with the user area in which that file logically resides. Physically, there is no distinction between user areas, because when a program requests file space the CP/M file management system allocates the first free block (i.e., the one with the lowest block number). The free block may have been released by erasure of a file, and may therefore be sandwiched between blocks belonging to a file of a different user.

General Usage. The ZCPR3 resident commands and ZCPR3-specific utilities may use either the DU form or the DIR form to identify a directory whenever the simpler D: form would be used under CP/M; the D part of the DU form is optional if the desired directory is on the currently logged-in disk, and the U part is optional if it is the same as that of the current directory. For instance, if the user is logged into disk B, user 5, the DU reference "A:" refers to disk A, user 5, and the DU reference "10:" refers to disk B, user 10. References such as "C31:" completely specify a particular disk and user area, (in this case, disk C, user area 31). The DIR form may be used instead of the DU form provided that a name has previously been defined for the target directory. For example, if the name ROOT is assigned to refer to disk A, user 15, then the ZCPR3 commands like "TYPE ROOT:MYFILE.TXT" and "DIR ROOT:" reference files on disk A, user area 15.

The user can be logged into any directory on any disk and readily work with files in any other directory on any disk. Just as the CP/M user can prefix a COM file with a disk letter in order to temporarily log into another disk and extract that file from it, so the ZCPR3 user can prefix a COM file with a DU or a DIR form.

In summary, the DU and DIR forms of directory reference can be employed in three basic ways:

1. To reference a directory location for a file, as in commands like "TYPE A15:MYFILE.TXT" or "DIR ROOT:"
2. To reference a directory location from which to extract a COM file, as in commands like "ROOT:MYPROG PARAMS"
3. To log into a directory, as in commands like "B7:", "12:", "C:", and "ROOT:"

Advantages of the DIR Form . The DIR form offers several additional features. Chief among these is that each named directory may also have a password associated with it. If a password is specified, any reference to such a directory by the user results in the user being prompted for the password. Should the user enter an invalid password, access is denied and the directory reference is changed to his current directory. If no password is specified, access is unrestricted.

The DIR form is more easily remembered than the DU form. A directory named "ASM" is much more easily identified as containing assembly language source code files than "B7."

Finally, the DIR form provides a mechanism which to some degree supports transportability of software between systems. Programs can now look for directories by the name in which their overlays and other working files may be stored. To illustrate, the HELP command of ZCPR3 searches for a file named in its parameter list ("HELP ZCPR3" searches for the file ZCPR3.HLP). When the HELP command is issued, it searches along the command search path (discussed below) for the specified HLP file, and, if this search fails, it looks for the file in a directory named "HELP." One system may keep the directory named "HELP" on disk A, user area 16, whereas another system may keep "HELP" on disk B, user area 31. Regardless, the HELP utility will find the correct directory.

ZCPR3 Prompt

The ZCPR3 prompt usually tells the user what directory he is logged into. ZCPR3 can be configured to present any one of four prompt formats to the user:

1. The prompt may be displayed as ">", in which case no indication is given as to which directory the user is logged into.
2. The prompt may be displayed as "d>" or "du>", in which case just the disk or disk and user area are presented to the user. "C>" and "B7>" are examples of such a prompt.
3. The prompt may be displayed as "dir>", in which case only the directory name is presented to the user. The user need never concern himself with the DU form and can think of all of his directories mnemonically. "ROOT>" is an example of this prompt. If the current directory does not have a name, this prompt appears simply as ">".
4. The prompt may be displayed as "du:dir>", in which case all information is presented. "A15:ROOT>" is an example.

Some of the new ZCPR3 utilities are specifically designed to manipulate named directories. Among these are:

■ CD Log Into a Named Directory (like the DU: or DIR: commands, but far more is done)

- PWD Print Working Directory; this command lists the names of the directories accessible to the user
- MKDIR Make a Directory; create a new set of named directories or modify an existing set

All of the ZCPR3 resident commands and utilities respond to the ZCPR3 DU and DIR forms, but conventional CP/M programs do not. If a DU or DIR form is presented in the command line of a conventional CP/M program, the form is usually interpreted as the disk referenced by the form. For example, if the "ROOT" directory is disk A, user area 15, then "WS ROOT:MYFILE.TXT" will be read by the WordStar program to mean "WS A:MYFILE.TXT". Some programs, such as PIP, will not respond favorably to the DU and DIR forms. As a general rule, only the D form should be used in conjunction with non-ZCPR3 utilities.

Command Search Path

A *path* is a sequence of directories that are to be searched for a particular file in the order specified by the sequence. The directories in the path may be specified by absolute DU forms, by symbolic DU forms, by DIR forms, or by a mixture of all three. The sequence always starts with the directory into which the user is currently logged, and ends with a special directory (by convention called "ROOT"). The ROOT directory contains COM files–such as directory display utilities, telecommunication packages, editors, and copy utilities–that are frequently used throughout the system and are sometimes called by application programs.

Although the terms "root" and "path" are taken from UNIX and have some functional resemblance to their UNIX equivalents, it is important to note that ZCPR3 directories are *not* hierarchical in the sense that UNIX directories are. In UNIX, a directory is a file containing filenames and other information about the files–and these names may belong to text files, binary executable files, or command scripts; but they may also belong to files which are themselves subordinate directory files, so that a "tree" of directories and subdirectories is possible. The BDOS treats directories in a different way from files; thus, under CP/M and ZCPR3 all directories are on the same level and search paths are arbitrary–there is no physical tree structure. If any tree structures are to be established, they will have to be done in a "logical" fashion. By making directories visible or not visible (which is possible when only the DIR form is used), logical trees can be created. Logging into a directory via CD, for instance, may run an ST.COM file which loads a new set of directory names, suddenly establishing visibility to these directories (establishing a node of the tree). See the description of the CD command for more details.

The PATH command is used to define the search path desired by the user. The following examples all define exactly the same path, but use the different forms. At the start of the path, the user is logged into Disk B, User 5.

```
Absolute DU:   B5 B0 A5 A15
Symbolic DU:   $$ $0 A$ A15
DIR forms:     LETTERS WP SPELL ROOT
```

A dollar sign used in symbolic DU forms specifies "Current Disk" if it appears in the first position, or "Current User Number" if it appears in the second position. Use of the DIR forms assumes that all the names have previously been assigned to specific directories.

Command Search Processing

Command processing under CP/M is really quite simple:

1. Input and parse command line from user or file.
2. Determine if it is a CCP-resident command; if so, run the command.
3. Determine if the current disk and user area contains a COM file with a name that matches the command; if so, load it and run it.
4. Print error message if 2 and 3 fail.

ZCPR3 offers a much more sophisticated and flexible command processing facility. The ZCPR3 command search hierarchy can be expressed as follows (the command processors to which reference is made will be fully described later):

1. Input and parse the command line from the user, a running SUBMIT file ($$$.SUB), or a ZEX or ZEX-like input source (ZEX is a memory-based command file processor which can be thought of as a memory-based SUBMIT).
2. Check the current Flow Command Package (FCP) to see if it recognizes the command; if so, run the command through the FCP.
3. Check the current Flow State to see if it is TRUE; if so, continue; if not, flush the command and advance to the next one (step 1).
4. Check the current Resident Command Package (RCP) to see if it recognizes the command; if so, run the command through the RCP.
5. Check the ZCPR3-resident command table for the command; if found, run it within the ZCPR3 command processor.
6. Search along the command search path for a COM file which matches the verb in the command line, logging into the disks and user areas indicated in the path until either the bottom of the path is reached or the desired COM file is found; load and run the program, if found.
7. If an Extended Command Processor has been specified (at installation time), load it and pass the command line to it for execution
8. If steps 2-7 fail, invoke an Error Handler program if one has been installed; if no Error Handler has been installed, print an error message.

The ZCPR3 command processor follows these steps when it attempts to resolve a command line presented to it. Because steps 1-5 use memory-resident facilities, this procedure is quite fast, and the ZCPR3 user realizes a very reasonable response time from the system.

Command Sources

In a full ZCPR3 System, there are four places where commands can be found:

1. Within the ZCPR3 command processor itself
2. Within memory-based resident command packages
3. Within memory-based flow command packages

4. In the form of COM files on disk

Table 2-1: Comparison of ZCPR3 and CP/M CCP Resident Commands

Function	ZCPR3 Command	CCP Command
Display $DIR File Names	DIR DU:afn	DIR D:afn
Display $SYS File Names	DIR DU:afn S	No Equivalent
Display All File Names	DIR DU:afn A	No Equivalent
Erase Specified Files	ERA DU:afn	ERA D:afn
Erase with Verify	ERA DU:afn V	No Equivalent
Rename File	REN DU:ufn=ufn2	REN DU:ufn=ufn2
Rename Over Existing File	REN DU:ufn=ufn2	No Equivalent
Print File on Console Without Paging	TYPE DU:ufn P	TYPE D:ufn
Print File on Console With Paging	TYPE DU:ufn	No Equivalent
Print File on Printer	LIST DU:ufn	No Equivalent
Save Memory into File Without Overwrite Warning	No Equivalent	SAVE n D:ufn
Save Memory into File With Overwrite Warning	SAVE n DU:ufn	No Equivalent
Save Memory into File and Specify Size in Hex	SAVE nH DU:ufn	No Equivalent
Save Memory into File and Specify Number of Blocks	SAVE n DU:ufn S or SAVE nH DU:ufn S	No Equivalent
Load File Anywhere into Memory	GET adr DU:ufn	No Equivalent
Reexecute Last Transient Without Reloading It	GO params	No Equivalent
Call Subroutine Anywhere in Memory	JUMP adr	No Equivalent
Change Disk	D:	D:
Change User	U:	USER u
Change Disk and User at Same Time	DU: or DIR:	No Equivalent
Prefix Commands	D:, U:, DU:, DIR:	D:

These four areas are briefly described below; more detailed discussions of the commands themselves and how the command processors work are contained in a later chapter.

ZCPR3 Command Processor. Like the CP/M CCP, ZCPR3 contains some built-in commands. It can contain all of the CCP commands (except USER, which is not needed) and a few more, but all of the ZCPR3 resident commands are different because they have logical extensions that offer features not found in the CP/M resident commands. Table 2-1 compares the various resident command forms under the CP/M CCP and ZCPR3.

The Extended Command Processor (ECP) is a program that is run by the ZCPR3 Command Processor. The entire command line is passed to the ECP, so the ECP can see the line as it was intended to be executed. It can perform its own additional parsing and evaluation and then either resolve the command or pass a new command line back to ZCPR3 for another round of interpretation.

If the ECP is not found, the conventional error message is given, flagging the original command as being in error. If the ECP (usually named CMDRUN.COM) is found, then the entire command line is passed to it as though it had been run as a command in its own right. For instance, if the original command line was:

```
MASM MYPROG
```

and the file MASM.COM was not found but the Extended Command Processor CMDRUN was, then this would be equivalent to issuing the command:

```
CMDRUN MASM MYPROG'
```

The utility of this feature can be seen immediately. Imagine that the SUBMIT program were renamed to CMDRUN. Then the failure of a command would cause SUBMIT to run and attempt to run a command file.

With this feature in mind, two ZCPR3 transients are provided for use as Extended Command Processors. They are SUB and ZEX. SUB and ZEX are command file processors. SUB is like an enhanced SUBMIT, and ZEX is also like an enhanced SUBMIT but it places its executable text into memory and runs much faster.

Resident Command Packages. Commands may also reside within a *Resident Command Package* (RCP). An RCP is a file which contains one or more commands and is loaded into memory by the LDR utility of ZCPR3 for direct execution by the ZCPR3 command processor. Each command within an RCP looks and acts like a COM file, but instead of having to be loaded from disk each time it is executed, the command executes immediately from within its memory-based RCP. Several standard RCPs are provided in the ZCPR3 distribution files, and Table 2-2 lists some of these commands and their functions.

Resident command packages offer several advantages to the ZCPR3 user:

1. Disk space can be saved, because a number of small commands can be grouped together in one file and loaded as a group for execution.
2. Time can be saved, because RCP-based commands are memory-resident once their RCP has been loaded; therefore, no disk activity is involved in locating and loading them.

3. Some commands, such as those built into the CP/M CCP, which are normally included in the ZCPR3 command processor, can alternatively be placed into an RCP, thereby freeing up the command processor for more system-oriented functions.
4. There is usually more space available in an RCP than within the ZCPR3 command processor, so RCP-based commands can be larger and more powerful than their ZCPR3-based counterparts.
5. Commands residing within an RCP generally do not occupy any space in the Transient Program Area; thus, if debugging facilities are made RCP-resident, they can examine the TPA after a transient program has been executed in an undisturbed state.

Table 2-2. Commands Available in Resident Command Packages

Command	Function
CP	Copy a file
ECHO	Echo the command line tail to the console or printer; this is useful in message display and device programming
ERA	Erase files, but an inspect option is available which displays each file and allows the user to approve
LIST	Print a group of files on the printer
MU	Memory utility—this is a screen-oriented memory editor which allows the user to change any byte anywhere in memory and examine locations in memory with ease; the TPA is not affected by running MU, so the last transient program run can be examined
P	Peek into memory, producing a dump of memory; the TPA is not affected
POKE	Poke into memory, changing byte values at will
PROT	Set protection attributes for files
TYPE	Type a group of files on the console

Flow Command Packages. A Flow Command Package (FCP) is very similar in nature to an RCP—it is a package of commands that is loaded by LDR for execution directly from memory. Commands that control the *Flow State* of the system are typically stored here. Nine flow states may exist at any one time in a ZCPR3 system: the empty state (which is TRUE) and IF Levels 1 to 8. The ZCPR3 command processor is constantly aware of the current flow state of the system, and if this state is TRUE, the ZCPR3 command processor will allow any command to execute if at all possible. If the flow state is FALSE, however, only commands which reside within an FCP may be executed. The IF command is usually used to raise the user to the next flow state level and set its value to TRUE or FALSE as the result of testing some condition. Table 2-3 shows the commands that are usually placed within an FCP.

Table 2-3. Common FCP Commands

Command	Syntax	Function	Examples

IF	IF cond	Test the indicated condition and raise to the next level, setting the Flow State to TRUE if the condition is TRUE and FALSE otherwise	IF EXIST FILE
			IF EMPTY FILE
			IF NULL $1
			IF ~NULL $2

FI	FI cmt	Terminate the current Flow State and drop down to the previous level (same meaning as ENDIF in conventional terms)	FI end inner IF
ELSE	ELSE cmt	Toggle the current Flow State	ELSE do other
XIF	XIF cmt	Exit all IF Levels, dropping to the empty flow state (which is TRUE), if the current Flow State is TRUE; else do nothing	XIF done

Command (COM) Files on Disk. The fourth type of command recognized by ZCPR3 is the standard COM file. This is an executable binary image which runs at location 100H. Programs such as WordStar (WS.COM) and dBASE II (DBASE.COM) are implemented as COM files. ZCPR3 handles COM files in much the same way as CP/M does. The difference is that ZCPR3 actively searches for a COM file in all directories on the current search path, whereas CP/M searches only a specific disk and user area and then gives up. The ZCPR3 user can shorten the search by explicitly stating where the file resides, but with the command search path he does not have to.

The ZCPR3 command processor loads the buffers in low memory in a manner quite similar to the CP/M CCP, so a COM file loaded by ZCPR3 sees these buffers and parameters in the same way it would see them under CP/M. ZCPR3 also loads some special buffers with additional information (such as the user areas referenced for the first two file names in the command line) that is meaningful only to ZCPR3-specific utilities. However, a normal CP/M COM file will not notice these buffers or be affected by them.

Multiple-Command Lines

The multiple-command line feature adds much flexibility and versatility to the ZCPR3 System. Unlike CP/M, ZCPR3 allows the user to specify, on one line, a sequence of commands separated by a semicolon. For example:

```
A>B:;DIR A7:*.TXT;DIR C22:*.COM A;C7:;ERA *.COM;DIR
```

is a valid command line to ZCPR3. This feature buys the ZCPR3 user two important advantages:

1. A sequence of time-consuming commands can be issued at one time, and the user can leave the system and do something else until the sequence completes. A ZCPR3 utility (SAK), designed with this in mind, rings the console bell to alert the user when the command sequence completes or reaches a critical point.
2. One program can invoke another program by placing a command line into the multiple-command line buffer, setting a pointer to the first character of the command line, and returning to the operating system. ZCPR3 will then resume command line execution at the pointer location and run the command for the previous program. This feature makes aliases and shells (among other things) possible.

Shells

The command search hierarchy described above is fundamental to the ZCPR3 command processor itself, but ZCPR3 also supports the concept of a shell. A *shell* is a program that acts as an interface between the user and the ZCPR3 command processor. Shells can completely change the user's mode of interaction with a ZCPR3 system; their function is to lift the user to a higher level of abstraction which is further away from the details of the machine he is using and closer to the problem he wants to solve.

Several shells are provided with ZCPR3, each with a programming language and environment of its own. MENU and VMENU are two such shells; they present menu displays to the user, allow him to select an item from a menu display with a single keystroke, and then build command lines based on this selection, passing these command lines to the ZCPR3 command processor for execution. When all commands in a sequence have been executed, the ZCPR3 command processor automatically reinvokes the shell, and the user finds himself back at his menu. All shells are documented in the chapter of this book which describes the commands.

The user never needs to know what commands are built and executed—all he needs to know is how to interpret the menu and strike a single character to select the function he desires. The shell hides from him all the details of implementation. Thus, an application coded in the command language associated with one of these shells can be run by a person with no technical knowledge except the little required to turn the machine on and select major functions.

Scripts

A *script* is a sequence of commands that may be stored in a disk file and executed as if they were a single command, merely by invoking the name of the file. A SUB file executed by the SUBMIT command of CP/M is an example of a command file of this type. However, because the ZCPR3 command processor is more powerful and flexible than the corresponding CP/M facilities, much more complex operations can be performed much more conveniently than could be done under CP/M.

There are many situations in which a user could find himself issuing the same sequence of commands, perhaps with minor variations, over and over again. Entering a complex command many times from the keyboard is not only boring, but leads to mistakes. These, in turn, can lead to wholly undesirable results that may not even be discovered until the last command terminates. At best, time is wasted editing the command line when a mistake is found before the RETURN key is hit. If each command is entered on a separate line of a script, mistakes are less likely and, better still, only a few keystrokes (the name of the script) are needed to invoke the sequence.

As a typical example the user may want to assemble a program (with the M80 assembler) and, if the assembler found no errors, link it (using the L80 linker) to create a COM file. To do this, the user might create the following script, in which $1 represents the program to be assembled and linked.

```
M80 =$1
< if this succeeds >
L80 $1,$1/N,A:SYSLIB/S,/U,/E
ERA *.REL
```

Since this set of commands could well be in a file ASM80.SUB to be executed by the CP/M command "A>SUBMIT ASM80", you might well ask "Why call it a script instead of a submit file, and why make a fuss about it?" The point is that this is not the only form of script, and that ZCPR3 offers the user another convenience to assist him in cases like this—the alias. An *alias* is a script that can be invoked as if it were a program in a standard COM file, merely by giving its name as a command. In the above example, the following alias could be created:

```
ASM80:
     M80 =$1;
     IF INPUT;
          L80 $1,$1/N,A:SYSLIB/S,/U,/E;
          ERA $1.REL;
     FI
```

The IF statement checks to see whether the M80 assembler actually generated a relocatable object file; the FI statement terminates the group of statements to be executed if that condition is true. Now, by issuing the command "ASM80 MYPROG", the script commands are run, with the following results:

1. The program MYPROG.MAC is assembled.

2. The sequence pauses to allow the user to see the results of the assembly; if it is successful, he may strike the RETURN key and allow commands 3 and 4 to execute; if it is not successful, the N option aborts commands 3 and 4.

3-4. If the user responded in the affirmative at step 2, the L80 linker is run and MYPROG.REL is erased.

5. The IF is terminated and the alias returns control to ZCPR3.

The great advantage of using an alias is that, since it is itself a command, the ZCPR3 command processors search for the alias name in all directories included in the current path; thus, there is no restriction as to where it resides, as there is with CP/M submit files.

Alias Applications. Aliases are employed in a ZCPR3 System in a variety of ways:

1. An alias is usually run on cold boot to execute a series of programs that initialize the system and establish an initial operating environment. By convention, this alias is called STARTUP and is stored in the multiple-command line buffer by the cold boot routine, for execution as soon as the system comes up. Using STARTUP, the environment descriptor, initial RCP and FCP, named directory file, and terminal description can be loaded automatically, and a MENU can then be invoked to allow the user to select what he wants to do from that point forward.
2. Another alias, by convention named ST, is used by the CD command. The command "CD DIR:" will invoke CD to log the user into the indicated directory, and, if the user has permission to enter this directory, CD looks there for the command ST.COM. If this command is found, CD runs it. ST can be an alias that initializes the user's environment for him by, for example, loading a new named directory file or bringing up a menu.
3. Commonly used command sequences can be stored in aliases, and those aliases can be stored in the ROOT directory (at the end of the command search path) for execution from any directory on the system.

ZEX Command File Processor

The ZEX command file processor is a special part of the ZCPR3 System. ZEX, which stands for *Z80 EXecutive,* is an integral part of the ZCPR3 System, and it provides a memory-based command file facility which is similar to SUBMIT but stores the commands in a memory buffer and executes them directly from this buffer. Unlike SUBMIT, ZEX is integrated with ZCPR3, and utilities can communicate directly with ZEX, looking at the commands it is about to issue and changing the command flow within ZEX.

ZEX provides information to the system through the Environment Descriptor, and a utility can read this information and find out where the next character ZEX is going to input comes from, where the first character of the command file is, and how to turn ZEX's command monitor on and off to control its operation. A program can then change these pointers and take control of the execution of commands via ZEX. The potentials unlocked by this capability are yet to be completely explored, but facilities like shells and the GOTO command are aided by this interaction with ZEX.

Environment Descriptor

Under CP/M, a few simple features of the design made it possible to transport binary files between different CP/M systems. This capability opened the door for the development of the CP/M world. Software engineers and programmers could write code that would run on any CP/M system, regardless of the hardware configuration or other capabilities of the user's system. A market was thereby created.

To remain compatible with CP/M, these simple features were retained in their entirety in the design of ZCPR3, with few changes. ZCPR3, however, offers the *environment descriptor* as an additional feature that opens many doors to the software developers. The environment descriptor specifies the characteristics of the user's CRT and printer, as well as the features available under a particular ZCPR3 system. This information makes it possible to transport programs such as screen-oriented editors from one ZCPR3 system to another at the binary level; the only special requirement is that a pointer to the target system's environment descriptor has to be installed in the program. The utility Z3INS performs this installation in a minimum amount of time.

Redirectable I/O

Redirectable Input/Output refers to the ability of the user to switch to different Input/Output devices during the course of a session. Optionally implemented through the I/O Byte, I/O under CP/M supports four logical devices, namely:

Console	(CON:)input (keyboard) and output (display)
Printer	(LST:) output-only
Reader	(RDR:)input-only
Punch	(PUN:)output-only

Each of these four logical devices may have any one of four physical devices assigned to it, allowing the user to work with as many as sixteen physical devices. Refer to the *CP/M 2.2 Alteration Guide* by Digital Research Inc. for information on device driver and parameter-passing requirements.

Device Assignment under CP/M. Under CP/M, the I/O Byte (at memory location 3) specifies the assignment of these devices. It is divided into four 2-bit fields, each field associated with a logical device; within a given field, each of the four possible values (00, 01, 10, 11) is associated with a particular physical device. Assignment of a physical device to a logical device can be done by STAT commands (e.g., "STAT CON:=CRT:"). Table 2-4 summarizes the logical and physical device assignments and mnemonics available through the I/O Byte. Table 2-5 lists the standard meanings of the physical device mnemonics.

Table 2-4. I/O Byte Assignments

Logical Device -->		LST:	PUN:	RDR:	CON:
Bit Position -->		7 6	5 4	3 2	1 0
Physical Assignment		----	----	----	----
0	00 Binary	TTY:	TTY:	TTY:	TTY:Physical
1	01 Binary	CRT:	PTP:	PTR:	CRT:Device
2	10 Binary	LPT:	UP1:	UR1:	BAT:Mnemomics
3	11 Binary	UL1:	UP2:	UR2:	UC1:

To make use of this structure, each logical device driver must, every time it is called, examine the value present in the I/O Byte field associated with that logical device, and use the value as an offset into a table to obtain the address of driver routine for the physical device specified in the I/O Byte. This process adds a good deal of code to the BIOS, and many systems do not implement it. Nevertheless, if you have, say, a dot matrix printer and a daisywheel printer, you can use either one as the list device by issuing the appropriate STAT command; without this feature, it would be necessary to change the physical cable connections to the output port.

Table 2-5. I/O Byte Devices

Physical Device	Typical Meaning
TTY:	Teletype
CRT:	Cathode Ray Tube Terminal
BAT:	Batch Processor (RDR=in, LST=out)
UC1:	User-Defined Console
PTR:	Paper Tape Reader
UR1:, UR2:	User-Defined Reader Devices
PTP:	Paper Tape Punch
UP1:, UP2:	User-Defined Punch Devices
LPT:	Line Printer
UL1:	User-Defined List Device

Device Assignment under ZCPR3. Under ZCPR3, a slightly different scheme for redirectable I/O has been implemented. The implementer, however, has the choice of continuing to use the CP/M scheme or switching to this new one.

When a ZCPR3 system cold boots, the BIOS loaded from the system tracks of the disk contains only a few primitive I/O drivers. Only the CRT as a console is enabled, and the reader, punch, and list devices are assigned to the CRT. No redirection is permitted at this time.

The BIOS is structured so that all the I/O entries in its jump table branch to a second jump table that is initialized by the Cold Boot Routine. This second jump table is placed on a page boundary at the beginning of a scratch area. It is in this scratch area that the physical device drivers reside. It is recommended that the size of this scratch area be approximately 2K bytes to allow space for all the drivers that might be required at any one time. Packages of I/O routines can then be loaded into it via the LDR utility, the jump table at the beginning of the scratch area being modified to point to the drivers that have been loaded.

This scheme for dynamically changing I/O device assignments has several advantages over the CP/M scheme. First, it requires less memory, not only because it eliminates the code and tables required by CP/M for sampling the I/O Byte and dispatching the I/O call to the appropriate driver, but also because only those I/O drivers currently in use need to be resident in memory—under CP/M, *all* drivers must be memory-resident, whether or not they are in use. Second, the assignment capabilities are more flexible. Under the CP/M scheme, the total number of devices is limited to sixteen, and of these no more than four can be dynamically assigned to any one logical device; under the ZCPR3 scheme you can have as many devices as you want in any mix, merely by loading the appropriate I/O driver packages from disk. Third, the ZCPR3 command processing facilities make it possible for application programs to call for a change of I/O device assignments, which is not easy under the CP/M scheme. Fourth, assignment of devices within an I/O package can still be made in a manner similar to the CP/M I/O byte, but ZCPR3 provides a mechanism to refer to devices by descriptive names (see the DEVICE and DEV commands).

Toolset

We have now examined the more important ideas that went into the design of ZCPR3, and have incidentally seen some examples of how they can make life easier for the user. But there are many components in ZCPR3—so many that it is easy to become confused by the complexity of their interaction. What thread can we find linking these components that will help us to understand and use ZCPR3 effectively?

Perhaps the most fruitful approach is to view the ZCPR3 System as a toolset from which the user can create his working environment, and change this environment to suit his changing needs. There are several classes of tools within the ZCPR3 System:

1. *Utilities* — tools that perform basic functions such as erasing files and displaying directories.
2. *Documentation* — tools that provide online help to the user.
3. *Programmer Aids* — tools that assist the user in debugging his programs.
4. *Shells* — tools that act as front-ends to the ZCPR3 command processor and provide a different type of interface between the user and the ZCPR3 System.
5. *Command File Processors* — tools that support the processing of files containing commands.

The following chapters describe the programs of the ZCPR3 Toolset. Chapter 3 contains brief descriptions of all the tools, in alphabetic order by program name. Chapters 4 through 7 describe some of the more complex tools in greater detail.

3 TOOLSET OF ZCPR3

This chapter describes in detail how to use the tools provided as part of the ZCPR3 system. The tools are described in alphabetic order of their names, for easy reference. Some of the more complex tools are very briefly described, with references to the later chapters in which they are more fully covered.

ALIAS (Version 1.1)

Syntax:

ALIAS <-- Define New Command
  or
ALIAS dir:ufn <-- Redefine Old Command

Function:

The ALIAS facility is the script expansion utility of ZCPR3. An Alias is a COM file, created by the ALIAS program, which contains one or more commands (separated by semicolons) to be placed in the command line buffer. When the Alias is invoked, parameters from the command line are implanted into the script contained within the Alias, and the resulting new command line is placed into the command line buffer and executed.

Options:

None.

Comments:

ZCPR3 MUST be implemented with an External Command Line Buffer in order for ALIAS to work.

The script of the internal command line supports parameter passing in a manner similar to ZEX and SUB. The variables $n (where 0 <= n <= 9) may be placed into the script, and the corresponding parameters will be substituted for the indicated variables. The variable $0 is the name of the Alias itself. The variable $* is the tail of the alias command line.

The current disk and user may be referenced by using the variables $D and $U. $D expands into the letter of the disk which was logged in at the time the Alias was expanded (the home disk), and $U expands into a number (in ASCII chars) representing the user area which was logged in at the time the Alias was expanded (the home user).

The ZCPR3 System file names are available to the Alias as the variables $Fn and $Nn, where 1 <= n <= 4. $F1 refers to FILENAME.TYP of System File 1, $N1 refers to FILENAME of System File 1, etc. Note that the SETFILE command is used to define the contents of the System file names.

'$$' expands into a single '$'.

Selected Error Messages:

"Ovfl" means that the expanded command line, combined with the remainder of the contents of the command line buffer, is too long to fit in the command line buffer.

Examples of Use:

```
ALIAS          -- define an Alias
ALIAS alias    -- display script of "alias.COM" and edit
```

Summary of Alias Variables:

The following table summarizes the variables which may be referenced within the body of an Alias.

$0	Name of Alias
$n	Parameter from Command Line (1 <= n <= 9)
$*	Tail of Command Line (everything after the verb)
$D	Home Disk
$U	Home User
$Fn	FILENAME.TYP of System File n (1 <= n <= 4)
$Nn	FILENAME of System File n
$$	The character $

Examples of Aliases

Case 1: The user is constantly issuing the following commands in the order indicated:

```
ASM myfile.BBZ
LOAD myfile
```

He can generalize it with the following Alias script:

```
ASM $1.BBZ;LOAD $1
```

If this Alias is named MYASM.COM, then typing "MYASM test" will be equivalent to "ASM test.BBZ;LOAD test".

Case 2: The user has two printers on his system. He is using redirectable I/O as implemented under ZCPR3, and he has two versions of Word Star (trademark, Micropro)–one for each printer. He can create an Alias containing the following script:

```
Script               Meaning
IF NEC=$2            Check to see if 2nd param is NEC
     DEV L NEC       If so, assign LST to NEC
     WSN $1             and run NEC version of WS
ELSE                 If not ...
```

```
        DEV L TTY           assign LST to TTY
        WST $1              and run TTY version of WS
   FI
```

If the Alias was named WSTAR, then "WSTAR myfile.txt" would be equivalent to "DEV L TTY;WST myfile.txt" and "WSTAR myfile.txt NEC" would be equivalent to "DEV L NEC;WSN myfile.txt".

CD (version 3.0)

Syntax:

CD dir:
or
CD du:

Function:

CD (Change Directory) is used to move from one directory to another by using the names or literal DU forms associated with the directories. CD first logs into the referenced directory, and, if there is a file named ST.COM in it, CD will log the user into the referenced directory and invoke ST.COM. If there is no file named ST.COM in the directory, CD will simply log the user in.

Options:

None.

Comments:

Under ZCPR3, there are two basic ways to log into a directory. One way is by using the DU: or DIR: prefix (e.g., B1:ASM>TEXT: or B1:ASM>C7:). The other way is by using CD (e.g., B1:ASM>CD TEXT: or B1:ASM>CD C7:).

The tradeoff is in user efficiency. If a directory is always used for a particular function, such as cataloging disks, CD may be preferred because it will not only log the user in but will also run ST.COM, which can set up his environment by running MENU or some other program or group of programs.

ST.COM is an Alias. The only purpose of ST is to load the multiple command line buffer with a command line when it is executed without any options. This command line may contain a reasonable number of commands which perform any desired set of functions.

In the ZCPR3 environment, good candidates for execution by running ST via CD include the following commands:

```
LDR file.NDR    <-- Set up a new directory environ
PATH path-exp   <-- Set up a new Command Search Path
MENU            <-- Invoke the MENU Preprocessor
```

```
     ECHO message   <-- Print a Message to the User
```

Using CD to log into a new directory can drastically change the user's environment. The names of the directories he can access can change (LDR changes the Memory-Based names), the command search path he uses can change, and he can even find himself in a MENU environment or other front-end instead of a ZCPR3 command environment.

Selected Error Messages:

"Command Line Overflow" means that there was not enough room in the command line to insert the command to invoke ST.COM.

Examples of Use:

```
     CD TEXT:          -- log into directory TEXT
```

CLEANDIR (version 1.0)

Syntax:

CLEANDIR dir: o
  or
CLEANDIR o

Function:

CLEANDIR "cleans" a physical disk directory. It loads the directory of the target disk into memory, sorts it alphabetically within each user area (ascending order by default), and writes it out to disk filling unused directory entries with E5.

A DIR: prefix is allowed, but only the disk reference is meaningful, so if "CLEANDIR ROOT:" is issued where ROOT: is A15:, then disk A is cleaned.

Options:

D	sort user areas and files in descending order

Comments:

CLEANDIR's sort on the disk directory has several advantages:

1. Utilities such as XDIR, which sort the disk directory after loading it, run faster since the directory is already sorted.
2. The possibility of recovering files by the UNERASE command is increased if CLEANDIR has been run on the directory recently before the files were erased. Note that any erased files absolutely cannot be recovered by UNERASE if CLEANDIR was run between the time they were erased and UNERASE was executed.
3. Use of DU3 to look at the directory is facilitated if the directory is already sorted by CLEANDIR.

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
CLEANDIR         -- clean current disk in ascending order
CLEANDIR D       -- clean current disk in descending order
CLEANDIR A: D    -- clean disk A in descending order
CLEANDIR TEXT:   -- clean the disk on which the directory
                    named TEXT is defined, in ascending order
```

CMD (version 1.0)

Syntax:
CMD cmd1;cmd2;...
or
CMD or CMD;cmd2;...

Function:
If CMD has an argument, it builds a new command line which begins with this argument and proceeds with the rest of the command line. For example, the first form "CMD cmd1;cmd2;..." is translated into "cmd1;cmd2;...". This allows sources such as MENU, VMENU, and VFILER to enter the "cmd1" variable from the user selection manually.

If CMD has no argument, the user is prompted for input, and this input is inserted into the command stream at the point of the CMD command. This is useful, for instance, when SHSET is used to define a shell sequence, and this sequence is to be exited at some time. For example, if the user typed in "mycmd" in response to the prompt, the second form "CMD;cmd2;..." is translated into "mycmd;cmd2;...".

CMD sets the error message whenever it runs. ERROR is turned on if no line was input to CMD.

Options:
None.

Comments:
CMD was built for use specifically with the SHSET command, although it may find other applications. The problem that CMD addresses is the case where the main program in the shell sequence knows nothing about shells, and it is desired to leave the sequence sometime. CMD provides this out. For instance, if MU3 is to be used as the main shell, then "SHSET MU3;CMD" will run MU3, allow the user to do what he wants, and then reenter MU3. If the user entered the command "SHCTRL POP" the shell stack would be popped and the "MU3;CMD" loop would be broken.

Additionally, CMD sets the ERROR message of ZCPR3, so that programs on down the line can determine whether input was made when CMD was run. If the user simply strikes a RETURN in response to the CMD prompt, an error is indicated. Tests can later be made, like IF ERROR, to check this and make the command flow change depending on the outcome.

Selected Error Messages:
Self-explanatory.

CMDRUN

Syntax:
CMDRUN text (this command is usually executed by ZCPR3 itself, not by the user)

Function:
CMDRUN is a sample Extended Command Processor. It is invoked automatically by ZCPR3 when the user command is not found via the command search path and no error handler is engaged.

The text which follows the verb is the text of the original command line.

Options:
None.

Comments:
CMDRUN is only a simple sample. It shows that the original command line is now available in the command tail buffer (at 80H). The formal CMDRUN which the user programs for his ZCPR3 System can extract the original command line from this buffer and manipulate it as desired.

Selected Error Messages:
None.

Examples of Use:
None.

COMMENT (version 2.0)

Syntax:
COMMENT

Function:
COMMENT allows the user to type as many lines as he wishes without them being processed by ZCPR3. It has no arguments.

If the user strikes a ^P, all subsequent lines he types will be printed on the printer.

Options:
None.

Comments:
COMMENT has two main applications in the ZCPR3 environment:

1. When the console displays are being recorded, COMMENT allows the user to type notes to a future reader without having to begin lines with a semicolon (;); all of these lines are clearly shown to be comments since they begin with the prompt "Comment>".
2. When console I/O is redirected to two different users, such as CRT and MODEM I/O in parallel, then COMMENT may be used to provide a simple mechanism for them to communicate; both users can type to each other without having any effect on the system (such as command processing).

COMMENT is aborted by striking a ^C as the first character of a line. Backspace and Delete both echo as Backspace, space, backspace, and ^X and ^U both erase the current line. ^P toggles printing.

If COMMENT is to be used to chat between two users, it is recommended that an over/out protocol be employed (as recommended for the UNIX* WRITE program). The first user types, and, when finished, terminates with the letter "o" for over. The second user types and signals completion the same way. Completion of the conversation may be signalled by "o+o" for over and out.

* [UNIX is a trademark of Bell Labs]

Selected Error Messages:
COMMENT generates no error messages.

Examples of Use:

```
Comment> Hi, Charlie, how's it going? o
Comment> Hi, Rick, fine ... and you? o
Comment> Fine, Charlie
Comment> Here is how I use XDIR -- let me do the typing
Comment> from now on; just watch, and I'll reenter COMMENT
Comment> when done ... here goes o+o
```

CP (SYSRCP)

Syntax:
```
CP dir:ufn1=dir:ufn2
```

Transient Counterpart:
MCOPY

Function:
CP copies one file from one directory to another or into the same directory under a different name.

Options:
None.

Comments:
This is a simple form of MCOPY. One major distinction is that CP can duplicate a file under a different name in the same directory; MCOPY cannot do this.

Selected Error Messages:
None.

Examples of Use:

```
CP f1.txt=f2.txt
CP a15:=f1.txt
CP a15:f2.txt=c5:f1.txt
```

CPSEL (version 1.0)

Syntax:
CPSEL cmd1,cmd2,...

Function:
CPSEL (CRT/Printer SELect) is a ZCPR3 utility that permits the user to select either CRT 0 or CRT 1, and Printer 0, 1, 2, or 3 from the current ZCPR3 Environment Descriptor. This dynamically changes the characteristics of the printer and CRT which are used by other ZCPR3 utilities, such as PRINT.

Options:
None.

Comments:
The commands may be any of the following:

Cc, c=0 or 1	Select CRT 0 or CRT 1
Pp, p=0-3	Select Printer 0-3
Dd, d=A (All), C (CRT), P (Printer)	Display Selection Values

The values affected by these selections include number of lines and columns on the CRT and number of lines and columns on the printer. The ability of the printer to form feed is also included.

Selected Error Messages:
None.

Examples of Use:

```
     CPSEL DA            -- display all devices
     CPSEL C1,P3,DA      -- Select CRT 1 and Printer 3;
                             display all devices when done
```

CRC (version 2.0)

Syntax:

CRC dir:afn1,dir:afn2, . . . o . . .

Function:

The CRC Check utility distributed with ZCPR3 uses the same CRC computation algorithm employed by Keith Petersen in his CRCK program, and the values come out the same.

The CRC Check utility computes the CRC values of a selected set of files and prints out the file names, their sizes (in Kb and number of records), and their CRC values in hexadecimal. A count of the number of lines of code (assuming text files) and a comment associated with each file can be optionally included. A list of ambiguous files names may be provided to CRC.

Options:

C	Comment Output; add comments to output listing on disk or printer
D	Disk Output; send output to the disk file CRC.CRC
I	Inspect Files and Approve Each File to be reported on before output is produced
L	Count Lines of Text and include in output (assume all files are text files)
P	Printer Output; send output to the printer

Comments:

CRC is useful when transferring files from one site to another. The CRC values of the files can be computed and listed at one site, transferred, and compared at the other site.

The L option adds the utility of tracking code size (in lines of code).

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
CRC *.MAC L -- Compute CRCs of all *.MAC files in the
               current directory; include lines-of-code
               count in display
CRC *.* DLC -- Compute CRCs of all files, include count of
               lines of text and comments on each file, and
```

```
                    write output to disk in file named CRC.CRC
```

DEV (version 1.0)

Syntax:

DEV command,command,...

Function:

DEV is a utility which manipulates the ZCPR3 redirectable I/O device drivers. It allows the user to display the names of the current devices and select them.

Unlike its counterpart DEVICE, DEV accepts all input from the command line and is not interactive.

Options:

None.

Comments:

Any DEV command may take the following forms. Only the first letters are significant in these commands:

```
    DISPLAY ALL <-- Display names of all devices
                    (D A is the same as DISPLAY ALL.)
    DISPLAY CON <-- Display consoles
    DISPLAY LST <-- Display printers
    DISPLAY PUN <-- Display punches
    DISPLAY RDR <-- Display readers
```

The full physical device name must be given in the following commands. Only the first character and the '=' are significant in the rest of the command.

CON:=name   LST:=name   PUN:=name   RDR:=name

C=name is the same as CON:=name.

Selected Error Messages:

"DEV NOT Initialized with I/O Base" means that this ZCPR3 System does not support Redirectable I/O.

"Redirection Not Supported" means that the loaded drivers in the I/O Package do not support redirection.

Examples of Use:

```
    DEV C=CRT,L=TTY   --assign CRT to CON: and TTY to LST:
```

DEVICE (version 1.0)

Syntax:

DEVICE <-- Enter Interactive Command Mode

Function:

DEVICE allows the user to interactively display the names of the available physical devices (actually, device drivers) which may be assigned to the logical devices. The user may also assign a physical device to a logical device by name.

Options:

None.

Comments:

DEVICE runs only in an interactive mode. It responds to single-character commands, completing the command names on the screen and prompting the user for further input.

The following commands are recognized by DEVICE:

D	Display Device Names
C	Select Console Device (CON:)
L	Select List Device (LST:)
P	Select Punch Device (PUN:)
R	Select Reader Device (RDR:)
X	Exit to ZCPR3 without prompting for confirmation

The Display Device Names command (D) asks the user for the devices to display. The possible responses are: A - All, C - Consoles, L - Lists, P - Punches, or R - Readers.

The C, L, P, and R commands assign devices *immediately*. The user types the name of the device to be assigned. If he strikes a return in response to the device name prompt, the command is aborted.

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
DEVICE -- invoke utility
```

DIFF (version 2.0)

Syntax:

DIFF dir:ufn o...

or

DIFF dir:ufn1,dir:ufn2 o...

Function:

DIFF compares two files. It can simply state if the two files are different (stopping immediately after the first difference is located) or it can list all of the differences between two files on a byte-for-byte basis. The form "DIFF dir:ufn o..." compares the file in the indicated directory with the file by the same name in the current directory. The form "DIFF dir:ufn1,dir:ufn2 o..." compares the two files indicated.

Options:

C	Compare Files Only and Stop at First Difference
M	Multiple Runs; when a comparison is complete, prompt the user for new disks, allow him to change disks, and then run the comparison again until the user says to stop

Comments:

If used to print out differences. Diff presents the following information to the user:

o Relative Offset from the beginning of the file
o Byte values in the two files:
- in Decimal
- in Hexadecimal
- in ASCII

Selected Error Messages:

"AFN Not Allowed" means that the user specified an ambiguous file name (one containing wild cards). Both file names must be unambiguous.

Examples of Use:

```
DIFF text:myfile.txt  -- prints differences between
                         MYFILE.TXT in TEXT: and MYFILE.TXT
                         in current directory
DIFF myfile.txt       -- compares MYFILE.TXT against itself
DIFF backup:myfile.txt mc
          --  compares MYFILE.TXT in BACKUP: with MYFILE.TXT
              in the current directory; stops as soon as
              a difference is found; when done, prompts
              the user to change disks (BACKUP could be a
              floppy, and this command is checking to see
              that all copies of MYFILE.TXT on several
              disks are the same)
```

DIR (version 1.0)

Syntax:
DIR dir:afn o...

Function:
DIR displays a formatted, alphabetized listing of the files in a disk directory.

Options:

A	Display both non-system and system files
S	Display only system files
T	Display files sorted by file type and name (sort by name and type is default)

Comments:
The syntax of DIR is not the same as that of XDIR and XD. DIR is designed to be small (only 2K) and fast, while providing more utility than the ZCPR3-resident or RCP-resident counterparts.

If the user wishes to use an option, the AFN must be filled with *.* – otherwise, the option will be interpreted as a file specification.

A slash used as a delimiter (DIR /A, for instance) automatically causes the built-in documentation to be displayed.

Selected Error Messages:
"Ovfl" means that there was not enough buffer space in the TPA to contain the disk directory.

Examples of Use:

```
DIR  -- displays all non-system files in the
        current directory in the following fashion:
                 1. Horizontal display
                 2. Sorted by file name and type

DIR *.* A  -- like above, but both non-system and
              system files are selected
```

DIR (CP-Resident)

Syntax:
DIR dir:afn o...

Transient Counterpart:
DIR, XD, XDIR

Function:
Display a disk directory to the user.

Options:

A	Select non-system and system files
S	Select system files only

Comments:
The DIR command is used to display the names of the files in the current directory without any bells or whistles (such as sorted output and file size information). It has three basic forms:

```
DIR DU:afn     -- Display $DIR File Names
DIR DU:afn S   -- Display $SYS File Names
DIR DU:afn A   -- Display All File Names
```

Selected Error Messages:
None.

Examples of Use:
(Assume the user is on disk B)

```
DIR               -- displays non-system files in
                     current directory
DIR *.* A         -- displays both non-system and
                     system files in current directory
DIR 4:            -- shows all non-system files on B4
DIR A4:*.HLP A    -- shows all files of type HLP on A4
DIR *.* S         -- shows all system files on B1
```

DIR (RCP-Resident, provided in SYS.RCP)

Syntax:
DIR dir:afn o...

Transient Counterpart:
DIR, XD, XDIR

Function:
Display a sorted disk directory to the user. The CP-Resident DIR does not sort the directory.

Options:

A	Select both non-system and system files

S Select only system files

Comments:
This command is better than the DIR in ZCPR3 in that it sorts the file listing. Horizontal display format is used.

Selected Error Messages:
None.

Examples of Use:

```
DIR -- displays all non-system files in the current
       directory in a sorted fashion (by file name and type)
DIR *.* A    -- displays both non-system and system files in
                      current directory
DIR ROOT:*.* A -- displays both non-system and system files
                      in ROOT directory
```

DPROG (version 1.0)

Syntax:

```
DPROG              <-- program from STD.DPG
DPROG filename     <-- program from filename.DPG
DPROG filename.typ <-- program from filename.typ
```

Function:
DPROG can be used to send any set of byte values in any desired sequence to the physical device assigned to the console, list, or punch logical device (e.g., a control string to change the font used by a dot matrix printer). DPROG reads the indicated or implied file after a path search, and transmits the byte sequence contained in the file to the selected device.

Options:
None.

Comments:
The file used to program the device is a conventional ASCII text file which contains four basic types of lines:

1. comment lines–lines in which the first non-blank character is a semicolon (;).

2. word definition lines–lines that begin with a dash (-) in column one followed immediately by a word.

3. DPROG command lines—lines beginning with a special DPROG command character (> or =).

4. output lines—any other line which does not conform to one of the three categories above; these lines generate the output sent to the device.

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
    DPROG         -- program from STD.DPG
    DPROG ASM     -- program from ASM.DPG
```

DPROG Programming:

DPROG is a 3K interpreter for a device programming language. This language allows the definition of *words* (symbols up to 16 characters long) that contain any combination of output format control instructions, text strings, and references to other words. Once a word has been defined, it can be named in an output line, and its definition (including embedded format controls) will be translated and sent to the console, printer, or punch device. Word references can be nested up to 128 levels deep. For example:

```
;
; Define Basic Words
;
-esc        (%c)              "\E"      ; the escape character
-ctrly                        "^Y"      ; the character control-Y
-test       (Char: %c %x %d\n)          ; character test format
-normal_form   (%c)                     ; normal output format

;
; Use Words
;
"This is a test\n" test "ABC" normal_form "\nEnd of Test"
```

Execution of the output line will cause the device to display/print the following:

```
   This is a test
   Char: A 41 65
   Char: B 42 66
   Char: C 43 67
   End of Test
```

The following 2-character escape sequences are output literally when used in format definitions, but are translated according to the current format definition when they appear in quoted strings.

^c	Define control character
\b	Backspace char
\d	Delete char (DEL)
\e	Escape char (ESC)
\l	New Line char (CRLF pair)
\n	Line Feed char (LF)
\r	Carriage Return char (CR)
\t	Tab char (TAB)
\#	Numeric value (forms are \d for decimal, \dH for hex, \dq for octal, \dB for binary:\1, \245, \33h, \0feH, \111b, \77q, etc)

Additionally, the format expression is of the form (<format text>) where <format text> can contain any character sequence as well as recognize the following output directives:

%c	Output chars as ASCII characters
%d	Output chars as floating decimal ASCII chars
%x	Output chars as 2 hex ASCII chars
%2	Output chars as 2 decimal ASCII chars
%3	Output chars as 3 decimal ASCII chars

Any text can surround these output directives, and each directive can be used as many times as desired in a format expression. Once a format expression is given, it is used until a new expression is defined. For example:

```
(%x %d ) "\12\10hA" (%c) "\12\10hA"
```

will output:

```
0C 12 10 16 41 65 ^L^PA
```

(where ^L and ^P are the ASCII control-L and control-P).

The user can direct output to the console, printer, or punch at any time (for programming the physical devices attached to these logical devices); there are debugging commands (pause to examine output, dump word definition table, dump format expression); and you can set up as many *.DPG files as you want for programming a variety of functions. DPROG is a true ZCPR3 utility and searches the current path for the *.DPG files. Thus, if *.DPG files are placed in the ROOT directory, they will be found from any directory on the system.

A word definition under DPROG takes the following form:

```
-word_symbol text_of_definition
```

where "-" is the first character in the line.

The following DPROG commands are available for debugging and other purposes:

Output Direction:

>C	Direct Output to Console
>L	Direct Output to List (Printer)
>P	Direct Output to Punch

Data Dump:

=	Dump both Word Table (Symbols) and Format
=F	Dump current Format Specification
=S	Dump current Word Table (Symbol Table)

Output Pause:

<	Pause and wait for user to strike key (^C will abort)

DPROG can be used within an alias, ZEX command file, or any other ZCPR3 environment. For instance, the following WordStar alias is reasonable:

```
IF NEC=$2
     DEV L NEC       <-- assign printer
     WSN $1          <-- run NEC version of WS
ELSE
     DEV L TTY       <-- assign printer
     DPROG CORRESP   <-- program printer for
                          correspondence
     WS $1           <-- run proper version of WS
FI
```

Listing 3-1 provides a clear example of how a .DPG file can be used by DPROG to program a Televideo 950 CRT.

Listing 3-1. Device Programming File ASM.DPG

```
;
; Programming Definitions for TVI 950 CRT Terminal
;
;
; Define Support Words
;
-esc                 (%c)    "\1bh"      ; The ESCAPE Character
;-esc                (%c)    "<ESC>"     ; The ESCAPE Character
```

```
-cr                 "\r"            ; <CR>
-ctrly              "^Y"            ; ^Y
-ctrlp              "^P"            ; ^P
;
; Define Functions
;
;
; Function Key:
;  FKEY   Fn|FnS    FKEY_FDX|FKEY_LOC|FKEY_HDX   "string"   CTRLY
;  FKEY             Prefix
;  Fn or FnS        Function Key Number or Number Shifted
;                        (F1 or F1S)
;  FKEY_FDX or      Full Duplex - Send to Computer Only
;  FKEY_LOC or      Local - Send to Terminal Only
;  FKEY_HDX         Half Duplex - Send to Computer and Term
;  "string"         Contents of Function Key
;  CTRLY            Terminator
;
-fkey               esc "|"
-f1                 "1"
-f1s                "<"
-f2                 "2"
-f2s                "="
-f3                 "3"
-f3s                ">"
-f4                 "4"
-f4s                "?"
-f5                 "5"
-f5s                "@"
-f6                 "6"
-f6s                "A"
-f7                 "7"
-f7s                "B"
-f8                 "8"
-f8s                "C"
-f9                 "9"
-f9s                "D"
-f10                ":"
-f10s               "E"
-f11                ";"
```

```
-flls              "F"
-fkey_fdx          "1"
-fkey_loc          "2"
-fkey_hdx          "3"
;
; Function Key String
;  Use: define MY_KEY, MY_XMIT, and MY_TEXT and then issue
;           the word FUNCTION_KEY
;
-my_key                  f1           ;select function key 1
-my_xmit                 fkey_fdx     ;select full duplex
-my_text                 ""           ;no text
-function_key            fkey my_key my_xmit my_text ctrly
;
; Cursor: C_OFF|C_BB|C_SB|C_BU|C_SU
;   C_OFF            No Cursor
;   C_BB             Blinking Block
;   C_SB             Steady Block
;   C_BU             Blinking Underline
;   C_SU             Steady Underline
;
-c_off                    esc ".0"
-c_bb                     esc ".1"
-c_sb                     esc ".2"
-c_bu                     esc ".3"
-c_su                     esc ".4"
;
; User Line: USER "string" CR
;   USER             Prefix
;
-user                      esc "f"
;
; Display User or Status Line: DISP_USER|DISP_STAT
;   DISP_STAT        Display Status Line
;   DISP_USER        Display User Line
;
-disp_user           esc "g"
-disp_stat           esc "h"
;
; Keyclick: CLICK_OFF|CLICK_ON
;   CLICK_OFF        Turn Off Keyclick
;   CLICK_ON         Turn On Keyclick
;
-click_off           esc "<"
-click_on            esc ">"
;
; Video: VIDEO_NORMAL|VIDEO_REVERSE
;   VIDEO_NORMAL     White on Black
;   VIDEO_REVERSE    Black on White
;
-video_normal        esc "d"
-video_reverse       esc "b"
;
; Screen: SCREEN_OFF|SCREEN_ON
;   SCREEN_OFF       Turn Screen Off
;   SCREEN_ON        Turn Screen On
;
-screen_off          esc "o"
-screen_on           esc "n"
;
; Clear Screen: CLS
;   CLS              Clear the Screen
;
-cls                         "^Z"
;
; End of TVI 950 Definitions
;
; The following commands actually program the user's terminal
;
screen_off
click_off   video_normal   disp_user      c_bu
;11111111112222222222333333333344444444445555555556
;123456789012345678901234567890123456789012345678901234567890
user
   "1-Dir  2-Edit  3-VFiler  4-MAC  5-M80  6-LASM        "
   "9-CLS 10-Scr 11-SLn" CR

-my_key        f1

-my_text       "xd\r"
```

```
function_key

-my_key         fls
-my_text        "xd "
function_key

;fkey fl        fkey_fdx         "xd\r"         ctrly
;fkey fls       fkey_fdx         "xd "            ctrly

fkey f2         fkey_fdx         "wm "            ctrly
fkey f3         fkey_fdx         "vfiler\r"     ctrly
fkey f3s        fkey_fdx         "vfiler "        ctrly
fkey f4         fkey_fdx         "zex mac "       ctrly
fkey f4s        fkey_fdx         "sub mac "       ctrly
fkey f5         fkey_fdx         "zex m80 "       ctrly
fkey f5s        fkey_fdx         "sub m80 "       ctrly
fkey f6         fkey_fdx         "zex lasm "      ctrly
fkey f6s        fkey_fdx         "sub lasm "      ctrly

fkey f9         fkey_loc         "^Z"             ctrly
fkey f10        fkey_loc         esc "8"          ctrly     ;smooth
fkey f10s       fkey_loc         esc "9"          ctrly     ;hard
fkey fll        fkey_loc         disp_stat        ctrly
fkey flls       fkey_loc         disp_user        ctrly

cls "TVI 950 Programmed: Assembler Configuration"

screen_on
```

DU/DIR Forms

Use the DU form, standing alone, to log into a different directory. This command has three basic formats:

Change Disk	D:
Change User	U:
Change both Disk and User	DU:

The DIR form standing alone may also be used to log into a different directory. The format is:

TEXT: (where "TEXT" is the name of the desired directory).

Named directories have passwords associated with them. Any attempt to access a named directory that has a non-blank password will cause the user to be prompted for the password. If he supplies an invalid password, access is denied and he remains in his current directory.

The DU: form may be disabled under ZCPR3, leaving only the DIR: form. This option allows directory access to be strictly controlled, since only named disk/user areas can be referenced and password protection is provided.

DU3 (version 1.0)

Syntax:

DU3 <-- enter DU3 at command level
or
DU3 command line <-- run initial DU3 command line

Function:

DU3 allows the user to manipulate the information on disk as easily as he can manipulate memory with DDT and MU3. DU3 completely opens up the disk to the user, so care should be taken when using this command.

Options:

None (command line)

Comments:

Chapter 8 describes how to use DU3.

Selected Error Messages:

Explained in Chapter 8.

Examples of Use:

See Chapter 8.

Invoking DU3:

DU3 is invoked by a command line of the form:

```
DU3 <text>
```

where <text> is any valid DU3 command sequence. If the first two characters of <text> are '/?', the built-in documentation is displayed, after which the user is returned to the operating system in accordance with the conventions employed by the Toolset.

Examples:

```
DU3 /?         -- Display Built-in Documentation
DU3 lb,g0,e    -- Execute commands to Log in Drive B,
                  goto Group 0, and enter editor at
                  the first Block of Group 0
```

ECHO (version 1.0)

Syntax:

ECHO text <-- send <text> to console
or
ECHO $text <-- send <text> to printer

Function:

ECHO echoes the text which follows it to the CON: or LST: devices. If the first non-blank character of this text is a '$', then ECHO sends its output to the LST: device.

Options:

None.

Comments:

The purpose of ECHO is two-fold:

1. To provide a convenient way to send messages to the console during the execution of a command file or command line; for example:

   ECHO Assembling;ASM myfile.BBZ;ECHO Loading;LOAD myfile as a single multiple command line will print the informative messages "ASSEMBLING" and "LINKING" during the respective commands

2. To provide a convenient way to send escape sequences to the CRT or printer; ECHO uses direct BIOS calls without any character translation, so sequences for programming intelligent devices can be issued by running echo and typing in those sequences; for example:

   ```
   ECHO ^Z
   ```

   will clear the CRT screen if ^Z is the Clear Screen character for the user's terminal, and

   ```
   ECHO $^L
   ```

   will form feed the printer (assuming that the printer responds to the form feed character).

NOTE:

Since the command input line editor capitalizes the command lines, all alphabetic characters are automatically capitalized when echoed.

Selected Error Messages:
ECHO generates no error messages

Examples of Use:

```
ECHO hello, world -- sends text "HELLO, WORLD" to console
```

ECHO (CP-Resident or RCP-Resident)

Transient Counterpart:
ECHO

Syntax:
ECHO text <-- send text to console
or
ECHO $text <-- send text to printer

Function:
CP- or RCP-resident ECHO commands behave in all respects like the transient counterpart previously described.

ELSE (from SYSFCP version 1.0)

Syntax:
ELSE anytext

Function:
If the current Flow State is TRUE, ELSE toggles it to FALSE.

If the current Flow State is FALSE and the previous IF Level is in a TRUE State, ELSE toggles the Flow State to TRUE. If the previous IF Level is in a FALSE State, ELSE does nothing.

Options:
None (any text may follow the verb ELSE).

Comments:
None.

Selected Error Messages:
No error messages are generated.

Examples of Use:

```
IF NEC=$1
      < statements >
```

```
ELSE
        < statements >
FI
```

ERA (CP-Resident)

Transient Counterpart:
ERASE

Syntax:
ERA afn <-- erase files
or
ERA afn V <-- erase files with verify

Function:
The ERA command erases the indicated file(s). If the Verify option is used, a list of all matching files is displayed and the user is prompted for confirmation. If the user confirms that erasure is desired, all files on the list are erased.

Options:

V	verify erasure

Comments:
Compare this ERA form to the RCP-Resident ERA form.

Selected Error Messages:
None - self-explanatory.

Examples of Use:

```
ERA b7:*.bak
ERA text:*.tmp V
```

ERA (RCP-Resident)

Syntax:
ERA afn <-- erase files
or
ERA afn I <-- inspect files before erasing

Transient Counterpart:
ERASE

Function:
The ERA command erases the indicated file(s). It is not able to erase Read/Only files, but it can erase System files. The name of each file is printed as it is erased. If the I (Inspect) option is used, ERA prints each matching file name and prompts the user for approval to erase; if approval is given, the file is erased, otherwise ERA leaves the file intact and proceeds to the next matching file name.

Options:

I	Inspect each file and request approval to erase.

Comments:
None.

Selected Error Messages:
None.

Examples of Use:

```
ERA *.TXT        -- erase all files matching *.TXT

ERA *.TXT I      -- display in turn all filenames matching
                    *.TXT; request approval to erase; if
                    approved, erase the file, otherwise
                    display next matching name.
```

ERASE (version 5.0)

Syntax:
ERASE dir:afn1,dir:afn2,... o...

Function:
ERASE erases files in the file list. If no option is selected, ERASE does not "see" system files, and requests permission to erase read-only files encountered, but read/write non-system files are unconditionally erased.

Options:

S	Erase system files encountered in the file list.
R	Erase Read/Only files in the list without asking the user for permission.
I	Inspect; ERASE displays the name of each file in the list and asks permission before erasing the file. If the user gives permission but ERASE discovers that the file is R/O with the R option off, it will request a second confirmation before performing the erasure.

Comments:
None.

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
ERASE text:*.txt,asm:*.tmp -- erase all .TXT files in the
                              TEXT: directory and all .TMP
                              files in the ASM: directory.

ERASE *.* I -- display all files in the current directory
               and request approval to erase each one.
```

ERROR1 (version 1.0)
ERROR2 (version 1.0)

Syntax:
ERROR1
or
ERROR2

Function:
ERROR1 and ERROR2 are error handlers. If the user runs either program, that program installs itself as the system error handler. If ZCPR3 cannot find the COM file referenced by a command verb, it invokes the installed Error Handler and passes the command line to it.

Options:
None.

Comments:
Both ERROR1 and ERROR2 display the error line to the user and provide him with four options as to how to process this line:

1 Replace the command in error with a new command
2 Skip the command in error and resume execution with the next command
3 Replace the entire command line
4 Throw away the command line and resume user control

Unlike ERROR1, ERROR2 is screen-oriented, using the Z3TCAP for support in order to provide a much "flashier" display.

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
ERROR1 -- install Error Handler Number 1
```

ERROR3 (version 1.0)
ERROR4 (version 1.0)

Syntax:
ERROR3
or
ERROR4

Function:
ERROR3 and ERROR4 are error handlers. If the user runs either program, that program installs itself as the system error handler. If ZCPR3 cannot find the COM file referenced by a command verb, it invokes the installed Error Handler and passes the command line to it.

Options:
None.

Comments:
ERROR3 displays the name of the COM file which was not found and then flushes the command line, returning control to the user.

ERROR4 prints the name of the COM file which was not found and then advances to the next command in the command line buffer. If there is no next command, user control is resumed. If there is a next command, command execution resumes there.

Selected Error Messages:
None.

Examples of Use:

```
ERROR3 -- install ERROR3
```

ERRORX (version 1.0)

Syntax:
ERRORX

Function:
ERRORX disengages the current error handler, leaving no error handler enabled. The default error control facility of ZCPR3 is now in effect; if an error occurs in the command line, the command line from that point forward is printed (followed by a '?').

Options:
None.

Comments:
None.

Selected Error Messages:
No error messages are generated by ERRORX.

Examples of Use:
```
ERRORX -- disengage any Error Handler currently enabled
```

FI (from SYSFCP 1.0)

Syntax:
FI anytext

Function:
FI terminates the current IF Level. If there is no current IF level, FI does nothing.

Options:
None (any text may follow the verb FI).

Comments:
None.

Selected Error Messages:
None.

Examples of Use:
```
IF EXIST MYFILE.ASM
      < statements >
ELSE
      < statements >
FI
```

FINDF (version 2.0)

Syntax:
FINDF afn1,afn2,... o

Function:
FINDF searches through all of the known disks and user areas for files matching any of the indicated file specifications.

Options:

S	Include System Files

Comments:
If the S option is omitted, FINDF will search only for Non-System files. Use of the S option causes FINDF to search for both System and Non-System files. The search begins at disk A and extends until FINDF encounters the last possible drive or a drive that is not loaded. All user areas (0 to 31) are examined.

FINDF displays the names of the files found, grouped by drive and user area.

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
FINDF xdir.com s  -- search all drives and user areas
                    (both system and non-system) for XDIR.COM

FINDF xd.com,help.hlp,myfile.txt
                -- search all Non-System files for XD.COM,
                   HELP.HLP, and MYFILE.TXT
```

GET (CP-Resident)

Syntax:
GET adr ufn

Function:
GET loads a file into memory starting at the specified page address. It requires two arguments: the number (assumed to be hexadecimal) of the 256-byte page in memory at which to start the load; and the name of the file.

Note that "adr" is a page number, not an absolute address. Thus, if adr=1, loading starts at location 100H; if adr=2d loading starts at location 2D00H, and so on.

Options:
None.

Comments:
None.

Selected Error Messages:
"TPA Full" means that the file has hit the top of the TPA in its load.

Examples of Use:

```
GET 40 myfile.bin -- load MYFILE.BIN into memory at 4000H
```

GO (CP-Resident or RCP-Resident)

Syntax:
GO parameters

Function:
The GO command reexecutes the last program loaded into the TPA without having to reload it.

The parameters are parsed in the same way as for any transient command, and the appropriate buffers are loaded by ZCPR3. After ZCPR3 has finished with the parsing and buffer loading, it "calls" the program loaded at 100H.

Options:
None.

Comments:
GO must not be used when a shell is active, since the shell comes into memory at 100H and overlays the last program loaded there. If GO is used, the probability is that the shell, not the last program, will be reinvoked.

Selected Error Messages:
None.

Examples of Use:

```
     XD
     GO ROOT: -- rerun XD with ROOT: as a parameter
```

GOTO (version 1.0)

Syntax:
GOTO label

Function:

GOTO is a ZCPR3 utility, designed to be run from within a ZEX command file that permits branching. It accepts only one argument, a label, which is defined within the ZEX file as a special comment of the form:

```
;=label
```

Any text which follows the "label" phrase is considered to be comment and is not processed.

Options:

None

Comments:

GOTO works correctly *only* if executed within a ZEX command file; otherwise GOTO will issue an error message.

Without the ZCPR3 Flow Control facility, GOTO would be of little value. With IF, however, GOTO is extremely useful in setting up loops and other flow-control constructs.

Selected Error Messages:

"ZEX Not Running" means that GOTO was executed from outside a ZEX command file.

"Label xxx Not Found — Aborting ZEX" means that the referenced label was not found within the command file, so ZEX execution is terminated.

Examples of Use:

ZEX Command File 1:

```
REG S1 0;note Register 1 = 0
;=start
XIF;note Exit all pending IFs
REG P1;note (Reg 1) = (Reg 1) + 1
ECHO Hello, World
IF ~1 3;note IF Register 1 <> 3
   GOTO START
FI
```

ZEX Command File 2:

```
M80 =$1;note Assemble File
; Strike ^C if Errors Exist - ^?
if ~nul $3;note IF there are 2 libs ...
    L80 $1/N,$1,$2/S,$3/S,SYSLIB/S,/U/E;note link all
    goto done
fi
```

```
if ~nul $2;note IF there is a 2nd arg ...
    L80 $1/N,$1,$2/S,SYSLIB/S,/U/E;note link lib $2
else;note        IF there is no 2nd arg ...
    L80 $1/N,$1,SYSLIB/S,/U/E;note link
;=done
fi
```

ZEX Command File 3:

```
if NEC=$2
     echo Terminal is NEC
     goto done
fi
if TTY=$2
     echo Terminal is TTY
     goto done
fi
if DIABLO=$2
     echo Terminal is Diablo
else
     echo Terminal is Undefined
fi
;=done
xif;note Exit all pending IFs
ws $1;note Edit file
```

HELP (version 5.0)

Syntax:

HELP <-- display HELP.HLP
 or
HELP filename.typ <-- display HELP file (if 'typ' omitted, HLP is used)

Function:

HELP displays HELP files in an interactive way to the user on his console CRT. It is also able to print selected screens or information sections on the printer.

Options:

None.

Comments:
See Chapter 4 for an overview of the HELP subsystem and a detailed description of the structure of HELP files.

Selected Error Messages:
See Overview of Help command, below.

Examples of Use:

```
HELP                    -- display HELP.HLP
HELP myfile             -- display myfile.HLP
HELP myfile.txt         -- display myfile.txt
```

Summary of User Commands under HELP

Cmd	Meaning
^	Go to Previous Level
.	Go to Root Level
M	Go to Menu of Current HELP File
S	Go to Start of information section
L	Go to Previous Frame
CR	(Carriage Return or Space) Go to Next Frame
^C	(Control-C) Return to ZCPR3
P	Print Current Screen Display (Frame) or information section

HELPCK (version 1.0)

Syntax:
HELPCK dir:ufn o <-- default file type is HLP

Function:
HELPCK checks the syntax of a HELP file. It analyzes the file, providing a variety of statistics and reporting on structural errors. Reports include a listing of the options if the file is user-indexed; this listing should be manually checked by the user to see that all options are included and no additional, hidden options exist.

Options:

P	Send report to Printer

Comments:
None.

Selected Error Messages:
Messages are self-explanatory.

Examples of Use:

```
      HELPCK myhelp -- report on MYHELP.HLP
      HELPCK myhelp P -- report on MYHELP.HLP on printer
```

HELPPR (version 1.0)

Syntax:
HELPPR afn1,afn2,... o...

Function:
HELPPR prints out a HELP file. It starts each information section on a new page and ignores form feeds (used to separate frames), so the data is presented in a sequential fashion.

Options:

H@head@	Heading Text that appears at the top of each page
I	Inspect Files (select) before printing
L	Number each line
Occ	Offset each line by cc spaces
Snn	Skip to page nn before beginning print
T	(If TIMELIB installed) turn off time display

Comments:
HELPPR is specifically designed to take advantage of the internal structure of HELP files to print the data in a logical manner. It is therefore preferred over PRINT when HELP files are to be printed.

Selected Error Messages:
Error messages are self-explanatory.

Examples of Use:

```
HELPPR myfile1,myfile2 o5 -- print myfile1.HLP and
                             myfile2.HLP offset by 5 spaces
                             on each line
HELPPR myfile s5 -- print myfile.HLP starting at
                    the 5th page
```

IF (version 1.1)

Syntax:

```
IF cond args
 or
IF ~cond args
```

Function:

IF tests the indicated condition to see if it is TRUE and, if so, sets the Flow State to TRUE (allowing the following commands to execute). If the condition is FALSE, the Flow State is set to FALSE (allowing only Flow Commands to execute).

Options:

Option	Meaning
T	TRUE (Flow State is Set to TRUE)
F	FALSE (Flow State is Set to FALSE)
EMPTY afn,...	If all files in the indicated list are EMPTY (size is 0K), then Flow State is Set to TRUE
ERROR	If the ZCPR3 Error Flag is Set, then Flow State is Set to TRUE
EXIST afn,...	If all files in the indicated list exist, then Flow State is Set to TRUE
INPUT	User input is enabled, and if the user strikes T, Y, <CR>, or <SP>, the Flow State is Set to TRUE
NULL afn	If there is no 'afn' (field is blank), then the Flow State is Set to TRUE
TCAP	If a Z3TCAP is installed, the Flow State is Set to TRUE
WHEEL	If the Wheel Byte is Set, the Flow State is Set to TRUE
reg value	If the indicated register (0-9) has the indicated value (0-255), the Flow State is Set to TRUE
afn1=afn2	If the two AFNs are identical in name (11 char FILENAME.TYP are same), the Flow State is Set to TRUE

Comments:

In all cases, if the indicated condition is TRUE, the Flow State is Set to TRUE; if the indicated condition is FALSE, the Flow State is Set to FALSE.

This command is invoked if the current Flow Command Package has the IF.COM facility enabled. If this is the case, whenever an IF command is issued, the FCP will load IF.COM from the ROOT directory into memory and execute it. The command tail is passed to IF.COM, and IF.COM acts as a conventional COM file from that point forward. All buffers are loaded correctly (FCBs at 5CH and 6CH, TBUFF at 80H, etc).

A leading tilde (~) character before a condition negates the effect of the condition. If the condition is FALSE, the Flow State is Set to TRUE, and vice-versa. Example:

```
    "IF ~T" is the same as "IF F"
    "IF ~NULL arg" is TRUE if 'arg' is non-blank
    "IF ~EXIST afn,..." is TRUE if 'afn,...' do NOT
       exist (AFN and AFN ... must each not exist)
```

For each condition given, only the first two characters are significant (eg, NU for NULL).

Selected Error Messages:

"No IF Condition Given" means that the condition expressed was not one of the valid conditions.

Examples of Use:

```
IF NULL $1
   --  if the indicated parameter (from within a SUBMIT
       or ZEX command file) is not provided, set the
       Flow State to TRUE

IF ~EXIST ZEX.ASM,ZEX.ZEX
   --  if any one of these files does not exist, the
       Flow State is set to TRUE

IF EXIST ZEX.ASM,ZEX.ZEX
   --  if any one of these files does not exist, the
       Flow State is set to FALSE (ie, all files
       must exist for a TRUE Flow State)

IF NEC=$1
   --  if the first passed parameter is the same as the
       file name "NEC.", then the Flow State is Set to TRUE

IF 5 5
   --  if Register 5 = 5, the Flow State is Set to TRUE
```

IF (FCP-Resident)

Syntax:

Same as for transient IF.

Function:

Same as for transient IF. The IF command described here is resident within SYSFCP 1.0 when the COMIF equate is set to FALSE.

Options:

Same as for transient IF except that the file lists are not permitted— only one ambiguous file name.

Comments:

This command is invoked if the current Flow Command Package has the IF.COM facility disabled. If this is the case, whenever an IF command is issued, the FCP will resolve it internally. The resolution of the IF command within the FCP itself is noticeably faster (approximately 0.5 to 1.5 seconds) than resolution by loading and executing IF.COM.

Each of the options of the Resident IF may be independently enabled or disabled. These options are installation-dependent, and the SHOW command will display the available options for any installation.

Selected Error Messages:

None.

Examples of Use:

See transient IF.

IFSTAT (version 1.0)

Syntax:

IFSTAT

Function:

IFSTAT displays the current IF level. IFSTAT will report a Level Number from 1 to 8 (IFs may be nested up to 8 levels deep) or will reply "No Active IF".

Options:

None.

Comments:

The Flow State *must* be TRUE for IFSTAT to run.

Selected Error Messages:

None.

Examples of Use:

```
    IFSTAT -- the current IF level is displayed
```

JUMP (CP-Resident)

Syntax:

JUMP address <-- branch to indicated hex address

Function:
JUMP can branch to any location in memory. It takes only one argument, which is the target address, specified as a 16-bit hexadecimal number (leading zeros may be omitted).

Options:
None.

Comments:
JUMP is useful for entering a PROM- or ROM-based routine, such as a monitor program.

JUMP 100 is the same as the GO command except that first FCB has '100' in it as a file name. The text following a "JUMP 100" instruction is parsed into the appropriate buffers as it normally would be.

Selected Error Messages:
None.

Examples of Use:

```
JUMP 100 -- "call" routine at 100H
JUMP F800 -- "call" the routine at 0F800H
```

LDR (version 1.0)

Syntax:
LDR ufn1,ufn2,...

Function:
LDR is a general-purpose System Segment loader for ZCPR3. It loads all of the ZCPR3 System Segments into their appropriate buffers, checking their format and content before approving and completing each load. Each System Segment is specified unambiguously.

Options:
None.

Comments:
The following System Segments are loaded into memory buffers by LDR:

*.ENV files	Environment Descriptors
*.FCP files	Flow Command Packages
*.IOP files	Input/Output Packages
*.NDR files	Named Directory Files
*.RCP files	Resident Command Packages
*.Z3T files	Z3TCAP Entries

The contents of each file to be loaded are read into a memory buffer and examined segment by segment. Segment-unique structural checks, based on the file type, are performed. If the checks are passed, the segment is copied into the correct memory buffer as determined by the data contained in the Environment Descriptor. If the checks are not passed, an error message is issued and the next file in the list is processed.

Since the Environment Descriptor (currently residing in its own memory buffer) provides the address at which to load a buffered system segment, it is important that the Environment Descriptor be the first segment loaded by LDR. An alternative to this procedure would be to make the BIOS initialize the Environment Descriptor on Cold Boot, but this would require a relatively large BIOS overhead (over 128 bytes for the initial Environment Descriptor).

When LDR loads an Environment Descriptor, it places it at the address installed in LDR during the ZCPR3 System installation. All other system Segments are loaded at locations specified by the Environment Descriptor currently residing in memory.

Selected Error Messages:

"filename.typ is not a Valid Type" means that the file type of the indicated file is not ENV, FCP, IOP, NDR, RCP, or Z3T.

"filename.typ Contains a Format Flaw" means that the structure of the indicated file was not correct.

Examples of Use:

```
LDR SYS.ENV,MYIO.IOP,MYCMDS.RCP,MYIFS.FCP
```

– load SYS.ENV, and, based on the data in this Environment Descriptor, load the I/O Package MYIO.IOP, the Resident Command Package MYCMDS.RCP, and the Flow Command Package MYIFS.FCP

```
LDR TERM1.Z3T
```

– replace the current Z3TCAP entry with TERM1.Z3T

LIST (CP-Resident)
LIST (RCP-Resident)

Syntax:
LIST ufn CP-resident and RCP-resident
LIST afn RCP-resident only

Transient Counterpart:
PRINT

Function:
LIST displays a file on the printer. No paging or formatting of any kind is performed.

Options:
None.

Comments:
The CP-Resident version of LIST accepts only an unambiguous file name. The RCP-Resident version accepts an ambiguous file name. The matching files are printed sequentially without any page breaks between files.

Selected Error Messages:
None.

Examples of Use:

```
LIST MYFILE.TXT -- print file on printer
LIST *.txt -- print all .TXT files in the
              current directory on the printer.
```

MCOPY (version 1.4)

Syntax:
MCOPY dir:=dir:afn1,afn2,dir:=dir:afn3,... o...

Function:
MCOPY is a file copy program designed for use under ZCPR3. It supports many features related specifically to the ZCPR3 System and is intimately tied into the ZCPR3 System.

The basic purpose of MCOPY is to copy files from one directory (disk/user area) to another under ZCPR3.

MCOPY only *copies* files; it does not rename them. This is a major difference between MCOPY and PIP. An attempt at renaming (e.g., MCOPY text:f1.txt=f2.txt) just copies F2.TXT into the TEXT: directory, but it is still named F2.TXT (F1.TXT is ignored).

Options:

E	Test for Existence of File on Destination and User Approves Copy before Copy is Done
I	User Approves Each File before Copy Begins
M	Multiple Copy (Repeat) Facility. This allows the user to backup several files to several disks by copying all the specified files, prompting the user for a new disk, and then copying the files again, continuing until

	the user tells MCOPY to stop
Q	Quiet Operation (No Activity Displays)
V	Verify Copies

The E option (Existence Test) looks in the destination directory to see if the file it is about to copy is already there. It then tells the user of its findings and asks him if he wants to go ahead with the copy. The user may elect to copy or not copy as he desires.

The I option (Inspect) displays all filenames that match the indicated source files, allowing the user to select which of them are to be copied.

The M (Multiple Copy) option pauses before starting the copy operation. During the pause, the user may abort the procedure or insert a disk into the source drive, the destination drive, or both, and then instruct MCOPY to proceed. After copying all of the indicated files, MCOPY pauses again, allowing the user to change disks again. This sequence continues until the user aborts the procedure.

The Q (Quiet) option turns off the MCOPY activity display. During "noisy" operation, MCOPY is constantly telling the user what it is doing. I feel that this is better than quietly having problems without the user knowing what is going on. MCOPY pays attention to the QUIET flag of ZCPR3, and the initial mode of MCOPY is set by this flag.

The V (Verify) option checks the copied file to insure that the copy is good. With this option engaged, MCOPY computes a CRC value of the source file during read operations; when copying is complete, MCOPY stores the computed CRC. MCOPY now reads the destination file and computes its CRC value. The two CRC values are compared; any difference indicates a copying error.

Comments:

MCOPY can also be used for making multiple backups, pausing between successive copy passes to allow the user to change disks. Once MCOPY has begun operations, the user need never warm boot the system after changing disks; MCOPY does that for him automatically.

If a destination directory is not specified, MCOPY looks for a directory named BACKUP: and copies to this directory if found. If there is no directory named BACKUP:, MCOPY will copy to B0: (this can be changed by reassembling MCOPY or by modifying it with DDT and then SAVEing the modified version).

In copying a file from one directory to another, MCOPY performs the following steps:

1. It logs into the source directory and scans for the files specified by the user.
2. It logs into the destination directory, determines if a copy of the file exists on the destination and, if so, deletes it.
3. MCOPY copies the file in the source directory into the destination directory.
4. MCOPY sets the attributes of the file in the destination directory to be the same as those in the source directory.

5. MCOPY optionally verifies both files by means of a CRC Check.

Selected Error Messages:

"NO Files – ^C to Abort" means that no files matching the indicated file spec were found.

"TPA Ovfl" means that there was not enough room in the Transient Program Area to support MCOPY.

"Disk Full" means that there is no more room on the destination disk for the files.

Examples of Use:

```
MCOPY FILE1.*,HELP:FILE2.HLP,TEMP:=TEST.TXT,HI.*
```

Files matching FILE1.* in the current directory are copied to BACKUP:, the file FILE2.HLP in directory HELP is copied to BACKUP:, the file TEST.TXT in the current directory is copied to TEMP:, and the files matching HI.* in the current directory are copied to TEMP:.

Once a DIR:= is encountered, the default destination is redefined. Encountering a different source, however, does not change the default source directory.

MENU (version 3.2)

Syntax:

MENU <-- run MENU.MNU
or
MENU ufn <-- run menu contained in file

Function:

MENU is the ZCPR3 menu front-end processor. It is a ZCPR3 Shell which reads a *.MNU file and processes commands from it.

Options:

None.

Comments:

MENU is a ZCPR3 Shell. See Chapter 5.

Selected Error Messages:

"No Command Line" means that the ZCPR3 System does not support an external Command Line Buffer. MENU must have this to run.

"No Shell Stack" means that the ZCPR3 System does not support a Shell Stack. MENU must have this to run.

"Shell Stack Full" means that the Shell Stack is full and MENU cannot push itself onto the stack.

"Shell Stack Entry Size" means that the Shell Stack elements are too short for MENU to store its parameters.

"TPA Full" means that there is not enough room in the TPA to load the *.MNU file.

Examples of Use:
See Chapter 5.

MKDIR (version 3.0)

Syntax:

MKDIR <-- enter utility
  or
MKDIR dir:ufn <-- enter utility and load NDR file

Function:
MKDIR creates Named Directory Files; these are disk files containing the mnemonic names and the disk/user areas with which they are associated. MKDIR is an editor. It provides a scratch area in which the user can set up a named directory, review it, edit it, and make any changes he wishes. When satisfied, the user can write it out to disk as a file or abort and throw it away.

Options:
None.

Comments:
See text under "Using MKDIR" in the section on Named Directories.

Selected Error Messages:
Self-explanatory.

Examples of Use:
See text under "Using MKDIR" in the section on Named Directories.

MU (RCP-Resident)

Syntax:

MU <-- invoke MU at 100H
  or
MU address <-- invoke MU at indicated address

Function:
MU is identical to MU3 in function, with the exception that the H command (Hexadecimal Calculator) is not supported. The difference between MU and MU3 is that MU executes as an RCP and MU3 executes as a transient. As an RCP, MU allows the user to examine the TPA without concern for side effects, so debugging

transients is simplified by this command. It is the only main command in the DEBUG.RCP provided in the ZCPR3 release.

Options:
None.

Comments:
MU may be invoked as a Shell by the SHSET command. The C command can be used from within MU to execute any desired command line, including the "SHCTRL POP" command which pops MU from the Shell Stack, terminating its operation as a Shell.

Comments pertaining to MU3 are generally applicable to MU as well.

Selected Error Messages:
None.

Examples of Use:

```
MU          <-- run MU
MU 0F400    <-- run MU but position at 0F400H
```

MU3 (version 1.0)

Syntax:
MU3 <-- Invoke MU3 pointing to ZCPR3 Env Desc
or
MU3 address <-- Invoke MU3 pointing to address (hex)

Function:
MU3 provides a screen-oriented editor which may be used to examine and modify memory at the user's discretion. It loads as a transient and runs from the TPA, starting at 100H. Since MU3 does not overlay the ZCPR3 CP, it allows the user to examine the operating system directly.

Options:
None.

Comments:
MU3 uses the ZCPR3 TCAP for support. WordStar cursor motion conventions apply, and the user's arrow keys may be active if they are specified in the TCAP entry.

All numeric input arguments (such as constants and addresses) are assumed to be hexadecimal by default. However, decimal numbers may be input by prefixing them with a '#' character. For instance, as an argument to the A (select address) command, the user may type 7d0 or #2000 to indicate memory location 7D0 hex (2000 decimal).

All commands are simple and self-explanatory. They include:

N	Enter Hex Numbers	T	Enter Text
A	Specify Address	+/	Next/Last Block
H	Hex Calculator	Arrows	Movement
^R	Refresh Screen	^C	Exit MU3
C	Enter ZCPR3 Command Line		

Selected Error Messages:

None.

Examples of Use:

```
MU3 F000 -- invoke MU3 and point to address 0F000H
```

The Commands of MU3

MU3 is quite simple to use and recognizes only a few commands. These commands are presented in a menu to the MU3 user as the program is running.

The MU3 display screen is formatted as indicated below:

```
                         MU3 Memory Editor

                                      Value

           Hexadecimal Memory Dump       ASCII Dump

-- Movement -- -------------- Operation ---------------
     ^E          A Enter Address          + Next Block
      ^          H Hex Calculator         - Last Block
 ^S <-+-> ^D     N Enter Hex Numbers     ^R Replot Screen
      v          T Enter Text            ^C Exit MU3
     ^X          C Enter Command Line
```

As the user moves the cursor about on the screen, the value in the upper right corner changes, indicating both the hex value and ASCII character representation of the byte being pointed to. Also, a cursor moves in the Hexadecimal Memory Dump region, indicating where the user is in the current 128-byte block.

Once the cursor is pointing to a desired byte, N or T commands may be used to change memory starting at the byte indicated by the cursor.

Movement Commands

The cursor may be moved around the screen using the WordStar cursor movement convention. If the arrow keys for the user's terminal are installed via the ZCPR3 TCAP, then they keys may also be used to move the cursor.

Select address At any time, the user may strike the letter A (case makes no difference) to select a different region of memory to view. MU3 will display 128 bytes of memory starting at the address given by the user.

Move One Block Forward (+) or Backward (-) The commands "+" and "-" move the display forward and backward, respectively, by one block (128 bytes). The movement is instantaneous, and the cursor is repositioned to the first byte in the new block.

Value Entry Commands

The N command is used to enter a group of hexadecimal numbers into memory starting at the address indicated by the cursor. The user is prompted for input, and he may then enter a series of hexadecimal values, separated by spaces. Entry terminates when the user strikes the RETURN key. Case is not significant. Any number prefixed with '#' is decimal.

For example, the following is a sample sequence of values which may be entered:

```
0 1f f3 ff 2c c3 0 2 3 4 #192 #255
```

The T command is used to enter text into memory starting at the address indicated by the cursor. The user is prompted for input, and he may then enter a string of characters. All characters input are significant. Entry terminates when the user strikes the RETURN key. Case is significant.

If the user wishes to embed a numeric value within a text string, the escape format <nn> is provided, where 'nn' is a hexadecimal value as shown above. The form <#nn> is also provided, where 'nn' in this case is a decimal value. The form '<<' inserts a single '<' character into memory.

For example, to enter the sequence '<this is a test>' followed by carriage return and line feed characters into memory starting at the cursor, the use would type:

```
<<this is a test><0d><#10>
```

'0d' is 0D hex and '#10' is 10 decimal or 0A hex. The leading '<<' translates into one '<'.

Other MU3 Commands

The C command allows the user to enter a command line for immediate execution by the ZCPR3 Command Processor. If MU3 is invoked as a shell via the SHSET command, this command provides an escape mechanism as well as a way to execute a command line from within MU3. The SHCTRL POP command will terminate the current shell on the shell stack.

The H command invokes a hexadecimal calculator. The user is asked to enter two hexadecimal numbers. The second number is added to and subtracted from the first number, the results being printed immediately. Again, decimal numbers may be entered by prefixing them with '#'.

The ^R command refreshes the screen for the user. This is handy if the screen was garbled in some way, such as by turning off the CRT.

The ^C command causes MU3 to exit to ZCPR3.

NOTE

Syntax:
NOTE anytext

Function:
NOTE is used to express comments. A line beginning with a semicolon (;) is a comment, and a command whose verb is NOTE (there may be many commands on one line, separated by semicolons) is a comment.

Options:
None.

Comments:
NOTE is available as a CP-Resident command, an RCP-Resident command, and as a transient. It is recommended that it be implemented as a CP-Resident command due to its very low cost and high utility in command files and aliases.

Selected Error Messages:
None.

Examples of Use:

```
    NOTE this is a comment
```

P (RCP-Resident)

Syntax:

```
P           <-- display next 256 bytes
   or
P addr        <-- display 256 bytes starting at addr
   or
P addr1 addr2  <-- display memory range
```

Transient Counterpart:
MU3

Function:
The P (PEEK) command allows the user to examine an area of memory. If the user simply types "P" with no address, the next 256 bytes of memory are displayed. If the user types "P address", 256 bytes of memory starting at the indicated address are displayed. If the user types "P addr1 addr2", memory in the range addr1 through addr2 is displayed.

Options:
None.

Comments:
PEEK does not modify the memory locations it displays.

Selected Error Messages:
None.

Examples of Use:

```
P 50c0          -- peek starting at 50C0H
P 4000 4fff     -- peek from 4000H to 4FFFH
```

PAGE (version 2.0)

Syntax:
PAGE dir:afn1,dir:afn2,... o...

Function:
The PAGE command lists one or more files to the console, one screen at a time. Unlike TYPE, PAGE knows the width and depth of the screen; thus, when wide listings (such as those produced by assemblers) generate wraparound lines, the wraparound lines are counted by PAGE and do not overflow the screen.

Options:

0-9	Set Character Print Speed Delay
I	Inspect files
L	Number lines
P	Disable pause at end of screen
Sn	Skip to page n and then begin

Comments:
While a file is being paged to the user, the user can strike one of the digits to vary the speed of the output dynamically. 0 is the fastest, 9 is the slowest. This option allows the user to scan selected portions of a file by running PAGE with the P option (so it does not stop when the screen is filled) and striking a digit from time to time to speed up over sections that are of no interest and slow down for sections he wants to read.

While the output is being directed to the screen, PAGE supports the following single-character commands to change the output display in various ways:

0 to 9	change speed (0=slowest, 9=fastest)
P or p	toggle pause when screen fills (the user can dynamically turn on and off the ability to delay when a screen fills)
^X	skip to next file

^C	abort to operating system
^S	pause output; any key will resume, and all of these commands (except ^S) will work

PAGE constantly looks for user input, so these commands can be issued at any time, including the intervals when PAGE has paused after filling the screen (its default) or has been halted by a ^S. Characters other than the valid command characters listed above are ignored.

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
PAGE *.txt p8        -- page all *.TXT files; begin with
                        paging off; set speed to 8.
PAGE myfile.txt s5   -- page MYFILE.TXT, starting at page 5.
```

PATH (version 3.0)

Syntax:
PATH
or
PATH path-expression

Function:
PATH allows the user to display the current path or set a new path. The display shows the path in three formats: Symbolic, Absolute (DU), and Named Directory (DIR). The path expression may intermix any of these formats as desired to express the new path.

Options:
None.

Comments:
PATH determines the address of the path with which it is going to work from the ZCPR3 Environment Descriptor.

The PATH command deals with path expressions—that is, a sequence of directory names which can be expressed as "ambiguous" DU forms, absolute DU forms, or Named Directory forms.

To illustrate, let's say that the user is logged into B1. The path "$0 A$ A0 ROOT" represents the sequence B0 to A1 to A0 to ROOT.

Selected Error Messages:
"Bad Expression at <text>" indicates there was an error in the path expression at the indicated point.

Examples of Use:

```
PATH $0 A$ A15 -- set path from current disk/user 0
                  to disk A/current user to disk A/user 15.
PATH A$ ROOT   -- set path from disk A/current user to ROOT:
```

POKE (RCP-Resident)

Syntax:

POKE address val1 val2 ... valn
or
POKE address "character string

Function:

The POKE command allows the user to change the content of memory. The user must specify an address to POKE; the values that follow may be numeric (hexadecimal assumed) or alphabetic (preceded by a double-quote mark). The two forms may be intermixed with leading values and a trailing character string:

```
POKE address val1 val2 ... valn "character string
```

Note, however, that once text input begins, further hex values are interpreted as text characters, so hex input is halted for the scope of the command.

Options:

None.

Comments:

There is no restriction on the memory locations that may be changed by the user. *Use of this command can be dangerous.*

Selected Error Messages:

None.

Examples of Use:

```
POKE f400 0 1 2
          -- place the values 0, 1, and 2 into memory
             starting at 0F400H
POKE f400 "this is a test
          -- place the ASCII values for the indicated
             characters into memory starting at 0F400H
POKE f400 1 2 3 "hello, world
          -- intermix hex values and text.
```

PRINT (version 2.0)

Syntax:

PRINT dir:afn1,dir:afn2,... o...

Function:

The PRINT command, like the LIST command, prints a file on the LST: device but offers many more options. It can print a heading, page the file, number the pages, number the lines, place a date/time stamp on the output, put the file name on the output, and perform yet other functions.

Options:

E	Exact Print (Expand Tabs, Form Feed, No Line or Page Numbers, No Heading)
F	Toggle default of file name display on page header (default is ON, so F turns off name display)
H<delim>text<delim>	Define Heading text to appear at the top of each page
I	Inspect Files (allow user to select files before printing begins)
L	Enable numbering of each line
M	Disable Multiple Run Flag (if multiple run is ON, then no "Set Top of Form" message appears for each file and PRINT moves from one file to another unattended); default is with Multiple Run ON
N	Disable numbering of each page
On	Offset each line. Move each line in the indicated number of chars from the left of the page
Sn	Start printing on page n
T	Toggle date/time stamp in the header of each page (a TIME subroutine must be assembled into PRINT to enable the date/time stamp feature)

Comments:

The characteristics of the printer are defined by the ZCPR3 Environment Descriptor. Such characteristics include the number of physical lines on a page, the number of lines of text on a page, the number of characters per line, and whether the printer can form feed or not. The Environment Descriptor contains options for four printers, and the CPSEL utility can be used to select the desired set of attributes.

The date/time stamp feature is very machine-dependent, and PRINT has to be reassembled to support it.

While PRINT is running, the following commands work:

^C	Abort and return to operating system
^X	Skip to top of next page and skip to next file

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
PRINT text:*.txt,*.txt o10n
      --  print all *.TXT files in the TEXT: directory
          and in the current directory; offset all
          lines by 10 columns, and do not number pages

PRINT myfile.txt s25
      --  print MYFILE.TXT starting at page 25
```

PROT (RCP-Resident)

Syntax:
PROT afn <-- set files to R/W and DIR
or
PROT afn R <-- set files to R/O and DIR
or
PROT afn S <-- set files to R/W and SYS
or
PROT afn RS <-- set files to R/O and SYS

Function:
The PROT command sets the standard CP/M file protection attributes (i.e., bit 7 of bytes 1 and 2 of the filetype) for a group of files. The R/O and System attributes may be set with the R and S options, respectively, given in any order as "RS" or "SR". Omission of one of these options toggles the opposite (i.e., omission of R makes the files R/W).

Transient Counterpart:
PROTECT

Options:

R	Select R/O [Read-Only] (R/W [Read-Write] otherwise)
S	Select SYS [System] (DIR [Non-System] otherwise)

Comments:
None.

Selected Error Messages:
None.

Examples of Use:

```
     PROT b7:*.com rs -- set all *.COM files in B7:
                             to R/O and SYS
     PROT text:*.txt -- set all *.TXT files in TEXT:
                             to R/W and DIR
```

PROTECT (version 2.0)

Syntax:

PROTECT dir:afn1,dir:afn2,... keys o...

Function:

The PROTECT command replaces the attribute set capabilities of the STAT transient and adds more flexibility. PROTECT allows the user to set/reset not only the read-only, system, and archive attribute bits (bit 7 of the three filetype characters), but also the tag bits (bit 7 of each of the eight characters in the filename) of a file or set of files. PROTECT always operates on both system and non-system files.

Options:

The KEYS are the attributes selected. The following keys are allowed:

R, S, A	Enable read/only, system, and archive bits
n	Set tag bit n (1 <= n <= 8)
I	Inspect
C	Control

Inspect Mode allows the user to look at each file before it is "protected" and permit or disallow the function to be performed on a case-by-case basis.

Control Mode allows the user to see the name of each file selected and manually set its attributes and tag bits. In response to the Control Mode prompt, the user can type in any combination of the letters A, R, S, and the digits 1-8 (the tag bits).

Comments:

None.

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
     PROTECT A4:*.COM,ROOT:*.COM RSI
```

Set the Attributes of all COM files in directories A4 and ROOT to Read/Only and System. Turn off the Archive attribute and all tag bits. Allow the user to inspect each file before the operation is performed.

```
    PROTECT ROOT:*.TXT
```

Clear all attributes and all tag bits of all files of type TXT in the directory named ROOT

```
    PROTECT A: C
```

Allow the user to manually set all attributes and tag bits of all files on Disk A in the current user

PWD (version 1.0)

Syntax:
PWD o

Function:
PWD displays the names of all named directories to the user. If the P (Password) option is included, then the passwords to the directories will be included in the display (but *only* if the Wheel byte is set).

Options:

P	Display Passwords

Comments:
If the Wheel byte is not implemented for a system, then its address is 0, where the JMP instruction to the Warm Boot routine in the BIOS is located. Since JMP is non-zero in value, the Wheel Byte is TRUE, and passwords will be displayed by PWD.

Selected Error Messages:
"Password Request Denied - Not Wheel" means that the P option was given but the Wheel Byte was not set, so passwords will not be displayed.

"Named Directory Buffer Not Available" means that named directories are not implemented.

Examples of Use:

```
PWD   - display named directories
PWD P -- display named directories and passwords (Since only
         P is valid as an option, the command could have been
         "PWD PASSWORD",the last 7 characters being ignored).
```

QUIET (version 1.0)

Syntax:
QUIET o

Function:
QUIET sets, resets, and displays the Quiet flag in the ZCPR3 Environment Descriptor.

Options:

D	Display the Quiet Flag
R	Reset (turn OFF) the Quiet Flag
S	Set (turn ON) the Quiet Flag

Comments:
Many ZCPR3 utilities read the Quiet flag in the ZCPR3 Environment Descriptor and respond accordingly. If the Quiet flag is set (ON), then certain informative messages are suppressed in order to cut down on the "noise" created by the command. If the Quiet flag is reset (OFF), then all messages are displayed.

Selected Error Messages:
No Error Messages are generated. An invalid command results in the Help screen being displayed.

Examples of Use:

```
QUIET R         -- turn OFF (reset) the Quiet flag
QUIET DISPLAY   -- Display the Quiet Flag
```

RECORD (version 3.0)

Syntax:
RECORD ON or OFF <-- Console Recording
or
RECORD ON or OFF PRINTER <-- Printer Recording

Function:
RECORD controls the Disk Output Facility of the Redirectable I/O Drivers. Copies of Console and Printer outputs can be created in disk files by the use of this facility, and it may be extended into a number of other applications as well.

Options:

ON	Enable Recording
OFF	Disable Recording
P	Reference Printer

Comments:

For RECORD to perform its function, it must be implemented in the redirectable I/O drivers. This is left as an exercise for the reader. The subroutines executed by the RECORD functions are implemented as simple RETurn instructions in the redirectable I/O drivers supplied with ZCPR3.

Four routines are accessed in the Redirectable I/O Driver package to control the RECORD function. They are:

COPEN	Enable Recording Console Output
LOPEN	Enable Recording List Output
CCLOSE	Disable Recording Console Output
LCLOSE	Disable Recording List Output

RECORD is indirectly tied into DEVICE. Invoking RECORD itself does not necessarily start the recording process immediately. To begin recording output onto disk files two functions must be performed:

1. RECORD has to turn the appropriate Driver ON
2. DEVICE has to select the appropriate Driver

Turning RECORD OFF during a recording session, closes the output file and makes it available for other uses. If RECORD is later turned ON, the output file may be deleted (if the same file is selected to record into). However, if a new device is selected while RECORD is ON (say, DEVICE CON:=CRT is issued), then recording is SUSPENDED (NOT turned off) until the recording device is selected again. With this capability, if it looks like the recording session is not going well, recording can be suspended, the problem fixed, and then recording can be resumed.

This combined system of DEVICE and RECORD provides a flexible output recording system. In addition, the output recording need not necessarily go to a disk file. It could be set up to send CON: output to the CRT and, say, a Remote Computer for processing.

Selected Error Messages:

"I/O Driver Address NOT Defined" means that there is no I/O Package in this ZCPR3 System.

"Disk Driver Module NOT Loaded" means that the I/O Package does not support the RECORD facility.

"No I/O Driver Module Loaded" means that LDR has not been run to load an *.IOP file.

Examples of Use:

```
RECORD ON      -- turn on recording for the console
RECORD ON P    -- turn on recording for the printer
```

REG (Transient, version 1.0)
REG (RCP-Resident)

Syntax:

REG Dr or REG r <-- Display Register r
REG Mr <-- Minus (Set Register r=r-1)
REG Pr <-- Plus (Set Register r=r+1)
REG Sr value <-- Set (Set Register r=value)

Function:

REG displays, adds 1 to, subtracts 1 from, or loads a value into, the indicated register. A ZCPR3 Register is a one-byte buffer (values may range from 0 to 255 decimal).

The value used to indicate a register is a character from '0' to '9'. The character '#' indicates all registers ("REG S# 0" stores 0 to all ten registers).

Options:

None.

Comments:

Registers are used for two purposes:

1. to support looping in ZEX command files (do something N times)
2. to pass parameter values from one program to another program which is executed later

REG has a counterpart command in the System Resident Command Package provided in the ZCPR3 distribution.

Selected Error Messages:

REG (Transient): "Invld Reg ID: c" means that the register indicated was not symbolized by '0' to '9' or '#'.

REG (RCP-Resident): None.

Examples of Use:

```
REG S0 4   -- reg 0 = 4
REG S5     -- reg 5 = 0
REG P      -- reg 0 = reg 0 + 1
REG P5     -- reg 5 = reg 5 + 1
REG M9     -- reg 9 = reg 9 - 1
REG D      -- show values
REG        -- show values
```

REN (CP-Resident)
REN (RCP-Resident)

Syntax:
REN dir:ufn1=ufn2

Function:
REN changes the name of a file. The format is newname=oldname (the standard CP/M order). The directory prefix on the first unambiguous file name indicates in which directory the original (and renamed) file resides. REN cannot rename a read/only file, but can rename a system file. If the new name (ufn1) already exists in the directory, the user is asked whether the file having that name should be deleted.

Transient Counterpart:
RENAME

Options:
None.

Comments:
None.

Selected Error Messages:
None.

Examples of Use:

```
REN newfile.txt=oldfile.txt -- rename OLDFILE.TXT to
                                NEWFILE.TXT in current dir
REN root:sys.rcp=sys1.rcp -- rename SYS1.RCP to SYS.RCP in ROOT:
```

RENAME (version 3.0)

Syntax:
RENAME dir:afn1=afno1,dir:afn2=afno2,... o...

Function:
RENAME changes the names of one or more files. Ambiguous file names and inspection are permitted.

Options:

C	Control Mode; manually specify each file name
I	inspect and approve each rename
S	Include System files

Comments:

The RENAME command is a brother to the REN resident command. There are many major differences, however:

- RENAME allows ambiguous file names to be used.
- RENAME supports an Inspect Mode, which presents the user with each name change and allows him to approve it before the change is made.
- RENAME supports a Control Mode, which presents the user with each file to be RENAMEd and allows him to enter the new name or to cancel the renaming.
- RENAME accepts a list of files.
- RENAME does not "see" System files unless told to
- RENAME can rename Read/Only files
- RENAME sets the attributes (R/O and SYS) of the new file names to be the same as those on the old file names

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
RENAME *.txt C -- rename all *.TXT files to something else;
                  display the file name to the user and
                  allow him to enter the new name

RENAME asm:*.mac=*.asm  -- rename all *.ASM files to *.MAC
```

SAK (version 2.0)

Syntax:

SAK o...

Function:

SAK (Strike Any Key) provides some simple utility functions, one of which is associated with the Multiple Command Line feature of ZCPR3. The functions are:

1. Allow the user to program a wait in a multiple command line until he instructs the system to continue.
2. Allow the user to abort a multiple command line.
3. Allow the user to program an interruptible delay in the execution of a multiple command line.

4. Provide a simple alarm for the user.

Options:

A	DO NOT allow the user to abort the command line
B	Ring the bell at the user's terminal occasionally
Pn	Pause n seconds and continue if no response by that time

Comments:

If no options are given, SAK waits for user input, and if the user strikes a ^C, then the multiple command line is aborted and control is returned to the user.

SAK is particularly useful if the user wishes to interject a delay in a multiple command line generated by a Menu. One such application is to display the time to the user, call his attention to it (via ECHO), give him a delay (via SAK), and then invoke dBASE II with an initializing command file.

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
SAK BP10    -- ring the bell occasionally and pause for 10
               seconds; if the user does not strike a command
               by that time (^C to abort), then resume
               command line execution with the next command.
```

SAVE (CP-Resident)

Syntax:

SAVE n ufn <-- save n pages (256 bytes) to file
or
SAVE n ufn S <-- save n sectors (128 bytes) to file

Function:

The SAVE command saves the contents of the TPA onto disk as a file. It accepts two arguments: a number and a file name. The file name may optionally be followed by the letter "S" to indicate that the number is the number of 128-byte Sectors (Blocks) to be saved. If this option letter is omitted, the number is assumed to be the number of 256-byte pages to be saved.

If the number, n, is followed by the suffix "H", as in "FH" or "2DH", then n is taken to be a hexadecimal value. If no suffix is given, n is assumed to be decimal. This hexadecimal option eliminates the need for conversion from the values supplied by debuggers such as DDT.

If the indicated file already exists, SAVE will ask the user if he wishes to erase it with the prompt "Erase ufn?".

Options:

S	Select sectors (128 bytes) instead of pages (256 bytes)

Comments:
None.

Selected Error Messages:
None.

Examples of Use:

```
SAVE 10 MYFILE.BIN   -- save 10 pages into MYFILE.BIN
SAVE 2FH HISFILE.BIN -- save 2F hex pages into HISFILE.BIN
```

SETFILE (version 1.0)

Syntax:
SETFILE n afn
or
SETFILE n

Function:
SETFILE sets the name of ZCPR3 System File n (where n is 1 to 4) to the indicated ambiguous file name. If no AFN is given, the current contents of the indicated System File are displayed. A file number is required.

Options:
None.

Comments:
ZCPR3 System Files are referenced by some of the ZCPR3 utilities, MENU and ALIAS in particular. SETFILE is a means by which the contents of these file name buffers are defined.

Selected Error Messages:
"Invalid File Name Number (not 1-4)" means that a valid file number was not given after the SETFILE verb (SETFILE n afn).

Examples of Use:

```
SETFILE 1 myfile.txt -- System File 1 is set to MYFILE.TXT
SETFILE 4            -- the name of System File 4 is displayed
```

SH (version 1.0)

Syntax:
SH

Function:
SH is a Named Variable Shell for ZCPR3. It prompts the user for a command line, performs an interpretation on the command line, and either executes the command line itself or passes the line on to the ZCPR3 Command Processor.

Options:
None.

Comments:
See Chapter 6, Shell Subsystem.

SHCTRL (version 1.0)

Syntax:
SHCTRL o

Function:
SHCTRL provides some control of the ZCPR3 Shell Stack from the command line. The contents of the Shell Stack can be displayed and popped one level or cleared completely.

Options:

C	Clear the Shell Stack (no Shell is in effect)
D	Display Shell Stack
P	Pop the Shell Stack (the current Shell is stopped and the next Shell on the stack is invoked)

Comments:
Only one option may be used in conjunction with the SHCTRL command. Any characters following this option are ignored.

SHCTRL is intended for use in situations where a directory change is desired and a Shell, such as MENU, is in execution. The Shell Stack can be popped, the directory change performed, and the original Shell explicitly reinvoked.

For additional information, see Chapter 6, Shell Subsystem.

Selected Error Messages:
None – Help is printed if invalid option is given.

Examples of Use:

```
SHCTRL P - pop the Shell Stack one level
SHCTRL D - display the contents of the Shell Stack
```

SHDEFINE (version 1.0)

Syntax:

```
SHDEFINE ufn  <-- define variables within file
   or
SHDEFINE      <-- define variables within SH.VAR
```

Function:

SHDEFINE allows the user to interactively display and edit the assignment of variables in a Shell Variable file. He may add, delete, and redefine Shell Variables as well as list all current definitions on the CRT or print them on the printer.

Options:

None.

Comments:

Use of SHDEFINE is explained within the program itself. It is menu-driven and the error and instructional messages are intended to be clear.

The most complex command is the E (for Edit) command. After issuing this command, the user is prompted for a variable name. If he gives the name of a variable not yet defined, the user is prompted for a definition, and the variable is so defined unless the user responded with just a RETURN at this point. If the name of the variable has already been defined, the user is asked if he wishes to delete (D) or redefine (R) the variable. Appropriate action is taken in response to the user input.

The user must be a Wheel to run the SHDEFINE command.

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
     SHDEFINE myvars -- define variables in MYVARS.VAR
```

SHFILE (version 1.0)

Syntax:

```
SHFILE    <-- display name of Shell Variable File
   or
SHFILE ufn <-- set name of Shell Variable File
```

Function:
SHFILE displays or sets the name of the Shell Variable File to be used by SH. SHFILE may be executed while SH is not running, if desired.

Options:
None.

Comments:
SHDEFINE and SHVAR define variables to be placed into Shell Variable Files. SHFILE defines which Shell Variable File will be used by SH when it executes.

When SH and SHVAR execute, the named variable file they deal with *must* reside in the ROOT directory.

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
SHFILE             -- display name of Shell Variable File
SHFILE myvars.var  -- define name of Shell Variable File
```

SHOW (version 1.0)

Syntax:
SHOW o

Function:
SHOW is the ZCPR3 Environment Display utility. SHOW generates numerous displays which include: details of the ZCPR3 Environment Descriptor, what system facilities are available, and the status of these facilities.

SHOW can be invoked as an Error Handler, in which case its Error Handler display can give the command line status and its other displays, such as memory examine, may prove useful to analyze the state of the ZCPR3 System.

Options:

E Install SHOW as an Error Handler (no SHOW displays are invoked)

Comments:
SHOW provides the following displays to the user:

```
1. Package Data                      3. ZCPR3 System
   -  Flow Command Package              - Environment Descriptor
   -  Input/Output Package              - Message Buffers
   -  Resident Command Package          - CRT and Printer Data
                                        -  System File Definitions
```

```
2. Environment Data
    -  Error Handler
    -  Memory Display Utility
    -  Named Directory Display
    -  Path Expression
    -  Shell Stack
```

SHOW is screen-oriented and will not function correctly without proper Z3TCAP support. The Environment Descriptor MUST be installed with a valid Z3TCAP entry.

Try it—you'll like it! SHOW is totally screen-oriented and will not function correctly without proper Z3TCAP support. If the user enters SHOW without proper support, the X command exits SHOW.

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
SHOW E -- install SHOW as an Error Handler
```

SHSET (version 1.0)

Syntax:
SHSET cmd1;cmd2;...

Function:
SHSET defines the commands which follow it as the command sequence to be placed on the top of the shell stack. It places this sequence there. Consequently, once SHSET is executed on a sequence of commands, these commands will be cycled thru time and time again until the shell stack is cleared or popped.

Options:
None.

Comments:
SHSET provides a simple mechanism through which the user can make any command sequence into a shell. Consequently, non-ZCPR3 programs, like MBASIC, DBASE II, WORD STAR, and others can become shells under ZCPR3.

When using SHSET, care should be taken to provide an exit from the shell (e.g., an exit to ZCPR3 and the system); unless this is done the system, once booted, will remain permanently in the application program. The CMD utility is sometimes useful for providing such an exit.

Selected Error Messages:
None – self-explanatory.

Examples of Use:

```
     SHSET MBASIC;CMD      -- define the sequence MBASIC;CMD
                              to be a shell
     SHSET WS              -- define WordStar to be a shell
```

SHVAR (version 1.0)

Syntax:

```
SHVAR        <-- list variables
     or
SHVAR var     <-- delete variable
     or
SHVAR var text <-- define/redefine variable
```

Function:
SHVAR can list all Shell Variables in the currently defined Shell Variable File, or can edit this file to delete or change one variable at a time. SHVAR is sometimes more convenient than SHDEFINE, which is intended for editing groups of variables in one sitting.

Options:
None.

Comments:
The user must be a Wheel to run SHVAR. SHVAR may be used whether SH is running or not. If a Shell Variable File is already defined to the ZCPR3 System, SHVAR uses this file. If one is not defined, SHVAR uses SH.VAR (always located in the ROOT directory).

Selected Error Messages:
Self-explanatory.

Examples of Use:

```
SHVAR                      -- list shell variables
SHVAR VAR2                 -- delete the variable VAR2
SHVAR VARX THIS IS A TEST  -- define VARX to "THIS IS A TEST"
```

SUB (version 3.0)

Syntax:

```
SUB or SUB //       <-- Print Help Message
    or
SUB /A Text        <-- Abort $$$.SUB File Processing at User's Discretion
    or
SUB /AB Text       <-- Same as /A but Ring Bell to Alert User
    or
SUB /I          <-- Enter Interactive Input Mode
    or
SUB filename params <-- As in Standard SUBMIT
```

Function:

SUB builds a command file on disk (named $$$.SUB). Each time ZCPR3 is ready for a command line, it looks for such a file and, if it finds one, extracts the next command from it.

SUB can be used in any situation where the user would normally use the CP/M SUBMIT command. XSUB will execute in conjunction with SUB if desired. SUB can also be used to sound alarms to the user.

Options:

A	Permit abort
AB	Permit Abort and Ring Bell
I	Interactive Input

Comments:

The "SUB /A" and "SUB /AB" forms allow the user to gracefully abort a $$$.SUB file. As under CP/M, entering ^C at the console can be used to abort such processing, but the /A form allows the luxury of starting a command stream and suspending it at one or more critical points. At each pause, the user can inspect what has happened, and then decide whether to proceed or not.

The "SUB /I" form allows the user to create a .SUB file without having to invoke an editor. If the user has a command stream he wants to execute immediately and doesn't care to do it again, he can use this option. In response, SUB allows him to enter his command stream (sorry, no parameter passing) a line at a time. When it is all entered (user enters an empty line), the $$$.SUB file is built and executed.

The "SUB filename params" form is identical to the form of the SUBMIT command supplied with CP/M. The "filename" specifies the name of the .SUB file to be executed, and the parameters are associated with the substitution variables $1, $2, etc. Up to 20 parameters may be specified. The sequence "$$" places a "$" into the command line, and the character sequence "^c" places the indicated control character into the command line (uparrow C places Control-C).

The SUB command may be nested into a $$$.SUB file. If a "SUB filename params" command is encountered in a $$$.SUB file, SUB runs, realizes that this has happened, and inserts the indicated command file, with parameter substitution, into the running command stream at the appropriate place. This may be nested as many levels deep as desired.

Under ZCPR3, if SUB is executed with the Multiple Command Line Facility invoked and more commands follow the SUB command, then the rest of the Multiple Command Line is inserted at the end of the generated $$$.SUB file.

SUB follows the ZCPR3 path when searching for the indicated command file. It is fully integrated into the ZCPR3 System and is able to employ the External Path and Multiple Command Line Buffer features of ZCPR3.

Examples of Use:

```
SUB cmdfile p1 p2 p3
   --  the file 'cmdfile.SUB' is processed, substituting
            'p1' for $1, 'p2' for $2, and 'p3' for $3
SUB /AB
   --  during the execution of a command file, this
      command causes the bell to ring at the console
      and the user is given a chance to abort execution
```

SUB Error Messages:

SUB provides a number of informative diagnostics to the user. In particular, when an error is encountered during processing of a command file, the user is informed of the line number at which the error occurred. The error messages presented by SUB are:

"Control Character" means that the ^c form was not followed by a letter A-Z

"Directory Full" means that there is no directory space for the $$$.SUB file

"Disk Full" means that there is no room to write the $$$.SUB file

"Line too Long" means that the current line in the command stream exceeds 128 bytes

"Memory Full" means that there is not enough memory in which to build the command stream to be placed into the $$$.SUB file

"Param" means that a parameter was referenced and none was given on the command line.

"Parameter" means that an Invalid Parameter was specified

"SUBMIT File Empty" means that the .SUB file specified in the command line was found to be empty

"SUBMIT File Not Found" means that the .SUB file specified in the command line could not be found along the ZCPR3 path

"Too Many Parameters" means that more than 20 parameters were on the command line

TCCHECK (version 1.0)

Syntax:

TCCHECK ufn <-- default file type is TCP
    or
TCCHECK <-- check Z3TCAP.TCP

Function:

TCCHECK checks a Z3TCAP.TCP file for valid format and reports any errors and statistical infórmation on it.

Options:

None.

Comments:

TCCHECK is intended to run in a non-installed environment (such as when thc user first rcceives ZCPR3), so the Z3TCAP.TCP file bcing chccked must be in the current directory.

Selected Error Messages:

Self-explanatory.

Examples of Use:

TCCHECK is used to check the Z3TCAP file for consistency. Its sole function is to ensure the validity of the Z3TCAP file and provide some statistics on it. The commands and responses displayed during a sample run are shown below.

```
B4:SCR2>tccheck
TCCHECK, Version 1.0  File Z3TCAP  .TCP Not Found - Aborting
       --  Note: Z3TCAP.TCP MUST be in the same directory
B4:SCR2>root:
A15:ROOT>tccheck
TCCHECK, Version 1.0
Z3TCAP File Check of Z3TCAP  .TCP Version 1.1
        File Checks with    44 Terminals Defined
```

TCMAKE (version 1.0)

Syntax:

TCMAKE ufn <-- default file type is Z3T

Function:

TCMAKE allows the user to interactively define the characteristics of his tcrminal and store this information in the file referenced. This file may then be loaded by the LDR utility.

Options:
None.

Comments:
None.

Selected Error Messages:
Self-explanatory.

Examples of Use:
TCMAKE is used to create a *.Z3T file. Once created, the ZCPR3 utility LDR can load it into memory at the proper location (command is "LDR filename.Z3T"). The commands and responses displayed during a sample run of TCMAKE are shown below.

```
B4:SCR2>tcmake //
TCMAKE, Version 1.0
TCMAKE - Create a Z3T File
Syntax:
        TCMAKE outfile -or- TCMAKE outfile.typ

where "outfile" is the file to be generated by
the execution of TCMAKE. If no file type is
given, a file type of Z3T is the default.

B4:SCR2>tcmake myterm2
TCMAKE, Version 1.0

        ** Z3TCAP Main Menu for File MYTERM2 .Z3T **

Define: 1. Clear Screen Sequence
        2. Cursor Motion Sequence
        3. Clear to End of Line Sequence
        4. Standout Mode Sequences
        5. Terminal Init/Deinit Sequences
        6. Arrow Keys
        7. Terminal Name

Status: S. Print Status (Definitions so far)

Exit:   X. Exit and Write File
        Q. Quit and Abort Program without Writing File
```

```
Command? 2

Cursor Motion Definition
 1. Timing Delay
 Enter Delay Time in Milliseconds: 5
 2. Enter R if Row/Column or C for Column/Row: R
 3. Enter Equation for Row: %+
 4. Enter Equation for Column: %+
 5. Enter Prefix Byte Sequence
  Char #1 - Type Char, .=Number, or <CR>=Done: Enter Number: 1bh
  Char #2 - Type Char, .=Number, or <CR>=Done: Char =
  Char #3 - Type Char, .=Number, or <CR>=Done:
 6. Enter Middle Byte Sequence
  Char #1 - Type Char, .=Number, or <CR>=Done:
 7. Enter Suffix Byte Sequence
  Char #1 - Type Char, .=Number, or <CR>=Done:

       ** Z3TCAP Main Menu for File MYTERM2 .Z3T **

Define: 1. Clear Screen Sequence
        2. Cursor Motion Sequence
        3. Clear to End of Line Sequence
        4. Standout Mode Sequences
        5. Terminal Init/Deinit Sequences
        6. Arrow Keys
        7. Terminal Name

Status: S. Print Status (Definitions so far)

Exit:   X. Exit and Write File
        Q. Quit and Abort Program without Writing File

Command? 6

Arrow Key Definition
 Your Terminal's Arrow Keys may be defined ONLY
 if they generate only one character each. If they
 do, type Y to continue. If not, type anything else.
        Define Arrow Keys (Y/N)? Y
Strike the Appropriate Arrow Key
```

```
 1. Arrow UP? ^K
 2. Arrow DOWN? ^V
 3. Arrow RIGHT? ^L
 4. Arrow LEFT? ^H

        ** Z3TCAP Main Menu for File MYTERM2 .Z3T **

Define: 1. Clear Screen Sequence
        2. Cursor Motion Sequence
        3. Clear to End of Line Sequence
        4. Standout Mode Sequences
        5. Terminal Init/Deinit Sequences
        6. Arrow Keys
        7. Terminal Name

Status: S. Print Status (Definitions so far)

Exit:   X. Exit and Write File
        Q. Quit and Abort Program without Writing File

Command? S

        ** Z3TCAP Status for File MYTERM2 .Z3T **

Review: 1. Clear Screen Definition
        2. Cursor Motion Definition
        3. Clear to End of Line Definition
        4. Standout Mode Definition
        5. Terminal Init/Deinit Definition
        6. Arrow Key Definition
        7. Terminal Name Definition

Exit:   X. Exit to Main Menu

Command? 1

Review of Clear Screen Definition
 1. Timing Delay = 0 Milliseconds
 2. Clear Screen Sequence:
  (1) ^[  1BH    (2) * 2AH
```

```
        Strike Any Key to Continue -

        ** Z3TCAP Status for File MYTERM2 .Z3T **

Review: 1. Clear Screen Definition
        2. Cursor Motion Definition
        3. Clear to End of Line Definition
        4. Standout Mode Definition
        5. Terminal Init/Deinit Definition
        6. Arrow Key Definition
        7. Terminal Name Definition

Exit: X. Exit to Main Menu

Command? 2

Review of Cursor Motion Data
 1. Timing Delay = 5 Milliseconds
 2. Row or Column First: R
 3. Row Equation:    -->%+ <--
 4. Column Equation: -->%+ <--
 5. Prefix Byte Sequence:
 (1) ^[   1BH    (2) = 3DH
 6. Middle Byte Sequence:
  -- Empty --
 7. Suffix Byte Sequence:
  -- Empty --
        Strike Any Key to Continue -

        ** Z3TCAP Status for File MYTERM2 .Z3T **

Review: 1. Clear Screen Definition
        2. Cursor Motion Definition
        3. Clear to End of Line Definition
        4. Standout Mode Definition
        5. Terminal Init/Deinit Definition
        6. Arrow Key Definition
        7. Terminal Name Definition

Exit:   X. Exit to Main Menu
```

```
Command? 6

Review of Arrow Key Definitions
 1. Arrow UP =    ^K
 2. Arrow DOWN =  ^V
 3. Arrow RIGHT = ^L
 4. Arrow LEFT =  ^H
        Strike Any Key to Continue -

        ** Z3TCAP Status for File MYTERM2 .Z3T **

Review: 1. Clear Screen Definition
        2. Cursor Motion Definition
        3. Clear to End of Line Definition
        4. Standout Mode Definition
        5. Terminal Init/Deinit Definition
        6. Arrow Key Definition
        7. Terminal Name Definition

Exit: X. Exit to Main Menu

Command? X

        ** Z3TCAP Main Menu for File MYTERM2 .Z3T **

Define: 1. Clear Screen Sequence
        2. Cursor Motion Sequence
        3. Clear to End of Line Sequence
        4. Standout Mode Sequences
        5. Terminal Init/Deinit Sequences
        6. Arrow Keys
        7. Terminal Name

Status: S. Print Status (Definitions so far)

Exit:   X. Exit and Write File
        Q. Quit and Abort Program without Writing File
Command? X
  Selected Terminal is: Rick's Terminal -- Confirm (Y/N)? Y
```

```
File MYTERM2 .Z3T Created
```

TCSELECT

Command:
TCSELECT 1.0

Syntax:
TCSELECT ufn <-- default file type is Z3T
or
TCSELECT <-- selection stored in Env Desc

Function:
TCSELECT allows the user to interactively review the contents of a Z3TCAP.TCP file and select a terminal from it. If an unambigous file name is specified in the command line, TCSELECT stores the selection into the indicated file. If no file name is given, TCSELECT stores the selection directly into the TCAP section of the memory-based Environment Descriptor.

Options:
None.

Comments:
None.

Selected Error Messages:
Self-explanatory.

Examples of Use:
TCSELECT is used to select a terminal from the standard Z3TCAP file. The selected terminal may be loaded directly into memory or a *.Z3T file may be created. If a *.Z3T file is created, the ZCPR3 utility LDR can load it into memory at the proper location (command is "LDR filename.Z3T").

Sample run of TCSELECT:

```
B4:SCR2>tcselect //
TCSELECT, Version 1.0
TCSELECT - Select Entry from Z3TCAP.TCP
Syntax:
        TCSELECT outfile  -or-  TCSELECT outfile.typ

where "outfile" is the file to be generated by
the execution of TCSELECT. If no file type is
given, a file type of Z3T is the default.
```

```
Syntax:
        TCSELECT
```

where this alternate form may be used to store the Z3TCAP entry for the selected terminal directly into the Z3 Environment Descriptor.

Example 1: Create MYTERM.TCP

```
B4:SCR2>tcselect myterm
TCSELECT, Version 1.0

** Terminal Menu 1 for Z3TCAP Version 1.1 **

A.  AA Ambassador             K.  Concept 100
B.  ADDS Consul 980           L.  Concept 108
C.  ADDS Regent 20            M.  CT82
D.  ADDS Viewpoint            N.  DEC VT52
E.  ADM 2                     O.  DEC VT100
F.  ADM 31                    P.  Dialogue 80
G.  ADM 3A                    Q.  Direct 800/A
H.  ADM 42                    R.  General Trm 100A
I.  Bantam 550                S.  Hazeltine 1420
J.  CDC 456                   T.  Hazeltine 1500

Enter Selection, + for Next, or ^C to Exit - +

** Terminal Menu 2 for Z3TCAP Version 1.1 **

A.  Hazeltine 1510            K.  P Elmer 1200
B.  Hazeltine 1520            L.  SOROC 120
C.  H19 (ANSI Mode)           M.  Super Bee
D.  H19 (Heath Mode)          N.  TAB 132
E.  HP 2621                   O.  Teleray 1061
F.  IBM 3101                  P.  Teleray 3800
G.  Micro Bee                 Q.  TTY 4424
H.  Microterm ACT IV          R.  TVI 912
I.  Microterm ACT V           S.  TVI 920
J.  P Elmer 1100              T.  TVI 950
```

```
Enter Selection, - for Last, + for Next, or ^C to Exit - +

** Terminal Menu 3 for Z3TCAP Version 1.1 **

A.  VC 404
B.  VC 415
C.  Visual 200
D.  WYSE 50

Enter Selection, - for Last, or ^C to Exit - -

** Terminal Menu 2 for Z3TCAP Version 1.1 **

A.  Hazeltine 1510         K.  P Elmer 1200
B.  Hazeltine 1520         L.  SOROC 120
C.  H19 (ANSI Mode)        M.  Super Bee
D.  H19 (Heath Mode)       N.  TAB 132
E.  HP 2621                O.  Teleray 1061
F.  IBM 3101               P.  Teleray 3800
G.  Micro Bee              Q.  TTY 4424
H.  Microterm ACT IV       R.  TVI 912
I.  Microterm ACT V        S.  TVI 920
J.  P Elmer 1100           T.  TVI 950

Enter Selection, - for Last, + for Next, or ^C to Exit - T

 Selected Terminal is: TVI 950 -- Confirm (Y/N)? N

** Terminal Menu 2 for Z3TCAP Version 1.1 **

A.  Hazeltine 1510         K.  P Elmer 1200
B.  Hazeltine 1520         L.  SOROC 120
C.  H19 (ANSI Mode)        M.  Super Bee
D.  H19 (Heath Mode)       N.  TAB 132
E.  HP 2621                O.  Teleray 1061
F.  IBM 3101               P.  Teleray 3800
G.  Micro Bee              Q.  TTY 4424
H.  Microterm ACT IV       R.  TVI 912
I.  Microterm ACT V        S.  TVI 920
```

```
J.  P Elmer 1100            T.  TVI 950

Enter Selection, - for Last, + for Next, or ^C to Exit - S

  Selected Terminal is: TVI 920 -- Confirm (Y/N)? Y

File MYTERM .Z3T Created
```

Example 2: Select terminal and store it in memory

```
B4:SCR2>tcselect
TCSELECT, Version 1.0
** Terminal Menu 1 for Z3TCAP Version 1.1 **

A.  AA Ambassador           K.  Concept 100
B.  ADDS Consul 980         L.  Concept 108
C.  ADDS Regent 20          M.  CT82
D.  ADDS Viewpoint          N.  DEC VT52
E.  ADM 2                   O.  DEC VT100
F.  ADM 31                  P.  Dialogue 80
G.  ADM 3A                  Q.  Direct 800/A
H.  ADM 42                  R.  General Trm 100A
I.  Bantam 550              S.  Hazeltine 1420
J.  CDC 456                 T.  Hazeltine 1500

Enter Selection, + for Next, or ^C to Exit - +

** Terminal Menu 2 for Z3TCAP Version 1.1 **

A.  Hazeltine 1510          K.  P Elmer 1200
B.  Hazeltine 1520          L.  SOROC 120
C.  H19 (ANSI Mode)         M.  Super Bee
D.  H19 (Heath Mode)        N.  TAB 132
E.  HP 2621                 O.  Teleray 1061
F.  IBM 3101                P.  Teleray 3800
G.  Micro Bee               Q.  TTY 4424
H.  Microterm ACT IV        R.  TVI 912
I.  Microterm ACT V         S.  TVI 920
J.  P Elmer 1100            T.  TVI 950
```

```
Enter Selection, - for Last, + for Next, or ^C to Exit - T

  Selected Terminal is: TVI 950 -- Confirm (Y/N)? Y

 ZCPR3 Environment Descriptor Loaded
```

TYPE (CP-Resident)
TYPE (RCP-Resident)

Syntax:

```
TYPE ufn <-- type file and page it
    or
TYPE ufn P <-- type file but do not page
```

Transient Counterpart:
PAGE

Function:
TYPE displays one or more files on the console. The file is paged (by default) or scrolled continuously if the P option is enabled. Flow control (XON/XOFF or ^S/^Q) is in effect.

Options:

P	disable paging

Comments:
The CP-Resident version of TYPE accepts only an unambiguous file name. The RCP-Resident version accepts an ambiguous file name, so several files can be displayed successively with a single command.

Selected Error Messages:
None.

Examples of Use:

```
TYPE MYFILE.TXT -- print file on console with paging
TYPE MYFILE.TXT P -- print file on console without paging
```

UNERASE (version 1.0)

Syntax:

UNERASE afn1,afn2,... o...

Function:

UNERASE recovers files which have been previously erased if it is possible to do so. As a rule, UNERASE has a much greater chance of success if it is used immediately after the files were erased. If anything is written to the disk after erasing a file, the directory entry or portions of the erased file may be overwritten.

Options:

L	List Erased Files Only (do not attempt recovery)
P	Pause for Disk Change and then try
Z	Place Recovered Files in User Area 0 (default is current user area)

Comments:

When ZCPR3 (and CP/M) erases a file, the information contained in the file and the directory reference to that information is not deleted immediately. Instead, the directory entry is simply marked as being deleted. As a result, by changing this delete mark back to a user number, the file is recovered.

After a file has been deleted, the directory entry and the blocks allocated to the deleted file all become available to the system for reallocation. Thus, a simple expansion of another file can overwrite one or more blocks previously allocated to the deleted file. Creation of a new file or expansion of an existing file that requires a new extent, may overwrite the directory entry of the deleted file. In all these cases, recovery of the deleted file is a complex process quite beyond the scope of UNERASE.

UNERASE is not always successful, however. If, for instance, the following events took place:

1 the files MYFILE.TXT, HISFILE.TXT, and T.TXT were created and then erased
2 a new file T.TXT was created and then erased
3 UNERASE T.TXT was issued–it is possible that both previous T.TXT files would be recovered, and a "weird" dual file named T.TXT would be in your directory

UNERASE prints the names of the files it is recovering, and, if a name appears two or more times, then these earlier files of the same name are being recovered. Those experienced with DU3 can probably identify the blocks allocated to the desired file, reconstruct the proper directory entry, and delete the incorrect directory entry. Others will simply have a corrupted directory which can only be restored by erasing T.TXT, thereby losing both files again. Occasional use of the command CLEANDIR can keep the directory clear to the point where UNERASE will function correctly most, if not all, of the time.

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
UNERASE myfile.txt,hisfile.txt
    --  try to recover MYFILE.TXT and HISFILE.TXT
UNERASE myfile.txt Z
    --  try to recover MYFILE.TXT and place it in User 0
UNERASE myfile.txt L
    --  see if MYFILE.TXT can be recovered (duplicates
        may also appear in this way)
```

VFILER (version 1.0)

Syntax:
VFILER <-- install VFILER as a shell

Function:
VFILER is a general-purpose, screen-oriented file manipulation utility. It allows the user to display, print, copy, rename, delete, and compute the size of files which are listed on the screen. The user can move his pointer about the list, using the WordStar cursor convention or, if the ZCPR3 TCAP for the user's terminal supports it, his own arrow keys may work.

Options:
None.

Comments:
For a detailed description, see Chapter 7.

Selected Error Messages:
Self-explanatory.

Examples of Use:
See Chapter 7.

VMENU (version 1.0)

Syntax:

```
VMENU           <-- run MENU.VMN on all files in dir
  or
VMENU afn       <-- run MENU.VMN on files selected by afn
  or
VMENU afn ufn   <-- run menu (ufn) on selected files
```

Function:
VMENU is the ZCPR3 menu front-end processor. It is a ZCPR3 Shell which reads a *.VMN file and processes commands from it.

Options:
None.

Comments:
For a detailed description, refer to Chapter 5, Menu Subsystem.

Selected Error Messages:
See Chapter 5.

Examples of Use:
See Chapter 5.

WHEEL (version 3.0)

Syntax:

```
WHEEL password S
   or
WHEEL password     <-- Set Wheel Byte
   or
WHEEL password R   <-- Reset Wheel Byte
   or
WHEEL /S or /R     <-- Enter Password without echo
```

Function:
The WHEEL command enables or disables certain privileged commands and command features. It does this by setting or resetting the Wheel Byte.

Options:

R	Reset Wheel Byte
S	Set Wheel Byte

Comments:
The Wheel Password is hard-coded into the WHEEL.COM file. It may be changed by DDT or reassembly.

WHEEL has an RCP counterpart, WHL, which may also be in effect. WHEEL and WHL can respond to different passwords.

Selected Error Messages:
"Invalid Password" means that the given password was not correct.

Examples of Use:

```
WHEEL mypass   -- set Wheel Byte if MYPASS is the
                  correct password
```

WHL (RCP-Resident)

Syntax:

WHL <-- make user non-privileged
or
WHL password <-- make user privileged
or
WHLQ <-- determine status

Transient Counterpart:

WHEEL

Function:

The WHL command is used to turn the Wheel Byte off (make the user non-privileged) or on (make the user privileged). The Wheel password is built into the RCP.

To find out the current status of the Wheel byte, use the WHLQ command (described below).

Options:

None.

Comments:

None.

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
WHL          -- make user non-privileged
WHL mypass   -- make user privileged if password is MYPASS
WHLQ         -- determine status of user (privileged or not)
```

XD (version 1.2)

Syntax:

XD dir:afn ooo...
or
XD /ooo...

Function:
XD displays a formatted, alphabetized listing of the contents of a disk directory.

Options:

Aa	Indicate attributes of files to be selected.
	a=A for All Files (System and Non-System)
	a=N for Non-System Files [default]
	a=S for System Files

Oo	Select Output Features
	o=A to Disable Display of File Attributes (R, S)
	o=F to Form Feed Printer when Display Done
	o=G to Group Files by Name and Type
	o=H to Display Files in Horizontal Format

P	Send Display to Printer
PF	Send Display to Printer with Trailing Form Feed

Comments:
It has been found that the more exotic features of XDIR (particularly the file scanner and disk output facility) are not used often in some environments. Because these features increase the size of XDIR.COM and reduce its execution speed, another version of XDIR.COM, called XD.COM, has been created.

XD is completely compatible with XDIR in terms of the options it accepts and how it operates. However, the file scanner and disk output facilities have been removed from XD. Hence, the D, F, and I options are not available. All other options of XDIR are retained in XD and perform in the same way. As a result, XD is smaller than XDIR (4K vs 8K), and therefore has a larger memory buffer in which to load files. In very large hard-disk systems, XDIR may not have enough buffer space to perform its functions, but XD almost certainly will.

The defaults of XD can be changed in the same fashion as those of XDIR; for details, see the "Comments" section of the XDIR command.

Selected Error Messages:
"TPA Error" means that there was not enough room in memory to load the disk directory.

Examples of Use:

```
XD -- Display the non-system files in the current
      directory in the following fashion:
                 1. sorted by file type and name
                 2. vertical format
                 3. R/O and SYS attributes included
XD *.COM AAOA -- display both non-system and system files
               which match *.COM in the following fashion:
                 1. sorted by file type and name
                 2. vertical format
```

```
                          3. no attributes included in display      ded
XD ROOT:*.COM AAOA -- same as above, but display only
                         files in directory named ROOT
```

XDIR (version 2.0)

Syntax:

XDIR dir:afn ooo...
or
XDIR /ooo...

Function:

XDIR displays a disk directory to the user and acts as a file name scanner.

Options:

Aa	Indicate attributes of files to be selected
	a=A for All Files (System and Non-System)
	a=N for Non-System Files [default]
	a=S for System Files

D	Send Output to Disk File XDIR.DIR

Ff	Enable a File Scanner Function
	f=L to Log File Names to FNAMES.DIR
	f=P to Print File Names Stored in FNAMES.DIR
	f=S to Scan Disk and Compare to FNAMES.DIR

I	Inspect Logged Files (use with FL option only)
N	Negate Selection of Files

Oo	Select Output Features
	o=A to Disable Display of File Attributes (R, S)
	o=F to Form Feed Printer when Display Done
	o=G to Group Files by Name and Type
	o=H to Display Files in Horizontal Format

P	Send Display to Printer
PF	Send Display to Printer with Trailing Form Feed
U	Select All User Areas

Comments:

XDIR is approximately 8K in size. The principal reasons to use XDIR in preference to the other directory display utilities are:

1 XDIR is able to send its output to disk.
2 XDIR provides the File Scanner Function.
3 XDIR can display all user areas of a disk.

If the default attributes of XDIR are not to the user's liking (e.g., if a listing by file name and type is preferred), there are three alternatives:

1 XDIR can be reassembled
2 The attributes can be patched via DDT (they start at the 6th byte from the front of the program)
3 An ALIAS can be created which selects the desired attributes for the user

Selected Error Messages:

"TPA Error" indicates memory overflow; there was not enough memory available in the TPA to load the disk directory

Examples of Use:

```
XDIR -- displays a listing of the non-system files in
        the current directory in the following fashion:
              1. Sort is by File Type and Name.
              2. Listing is in a Vertical Format.
              3. Attributes of Each File (R/O, SYS) shown.
              4. File Sizes in K.
              5. Total of Sizes of All Files.
XDIR /AA -- like the first example, but both non-system
           and system files are displayed.
XDIR /OH -- like the first example, but listing format
           is horizontal.
XDIR /OG -- like the first example, but sort is by file
           name and type.
XDIR *.COM /NFL -- the names of all non-system files
                  which do not match *.COM are stored on
                  disk in a file named FNAMES.DIR.
```

XDIR Summary

XDIR runs in two basic modes:

- a directory display utility
- as a file scanner utility

As a directory display utility, XDIR displays information about the files on a particular disk in all user areas or a particular user area. It provides the following information:

- Name of File
- Sizeof File (in Kb)
- Attributes of File (Read/Only or System)
- Sum of Sizes of All Files Displayed
- Total Number of Files on Disk
- Amount of Space Remaining on Disk

- What Disk and What User Area is being displayed

As a file scanner utility, it does the following:

- Logs a group of selected files to disk
- Prints the contents of such a log file
- Scans a log file and compares it with the files selected by the user, telling him what files are missing and what files are additional

XDIR is convenient to use, and contains many built-in features that provide flexibility in meeting the user's preferences. Some of these include:

- Named Directories may be specified.
- The file listing is alphabetized by file name and type or file type and name, depending on user preference.
- The file listing is organized vertically or horizontally, depending on user preference.
- Output may be sent to disk or printer as well as to the console.

XDIR Output Control

The Output Control options of XDIR are:

D	Send Output to Disk
Oo	Output Control OA - Toggle Display of File Attributes OF - Toggle Send of Form Feed with Print OG - Toggle File Grouping (name/type or type/name) OH - Toggle Format (Horizontal or Vertical)
P	Send Output to Printer
PF	Same as POF, which sends output to printer and does a form feed on completion

If P is specified, the output goes to the printer (LST: device) as well as to the console. If D is specified, the output goes to the file XDIR.DIR in the current directory. If XDIR.DIR already exists, it is replaced.

OA allows the user to display or suppress the file attributes field. This field, which follows the file size field, contains the letter R or the letter S, indicating, respectively, that the associated file is Read/Only or a System file. If R is not present, the file is Read/Write, and if S is not present, the file is Directory (i.e., non-system).

OF allows the user to select an automatic form feed when the directory display is sent to the printer. If printer output is selected (P option) AND the form feed flag is ON, then the last line of the printout will be followed by a form feed character. Many printers respond to this character by advancing to the top of the next page. Note: the special form PF is provided to act the same as P (for turning on printer output) and OF (to toggle the form feed function).

OG switches the display order from file name and type to file type and name, and vice versa. If the display is by file name and type, then files having the same name are grouped and AA.TXT comes before **BB**.COM. If the display is by file type and name, then files of the same type are grouped and BB.COM comes before AA.TXT.

OH allows the user to switch from vertical to horizontal format and vice versa. The display is divided into three columns; vertical format lists files first down column 1, then down column 2, and finally down column 3. Horizontal format lists the files sequentially across columns 1, 2, and 3 in row A, then columns 1, 2, and 3 in row B, and so on.

XDIR provides such a wide variety of output displays that the user is advised to experiment with the various XDIR options to see which format he prefers. The default settings for the various options can be changed to generate the preferred format by intelligent use of DDT or by reassembly of the XDIR.MAC source.

XDIR File Selection

The following options (and the DIR: field) control file selection:

```
Aa Select attributes of the files to be displayed
     a=S for system files
     a=N for directory (non-system) files
     a=A for all files (both system and directory))
N   Negate selection; select those files which do NOT match the ambiguous
      file name
U   Select all user areas
```

The A option selects the attributes of the files to be displayed. AA displays both non-system and system files; AS displays only system files; and AN displays only non-system files.

The N option selects all files which do not match the ambiguous file name. The scope of the N option is within the attributes selected, so if the attributes are AS, only system files are shown.

The U option selects all user areas on the specified or default drive. On the display, each file name is preceded by the user area in which it resides.

XDIR File Name Scanner

The options of XDIR which deal with the file name buffer facility are:

```
Ff Engage file name buffer facility
     f=L to log file names to disk
     f=P to print names logged to disk
     f=S to scan disk for file names and compare to log
I   Inspect files selected by FL option
```

The FL option writes the user numbers and file names of the selected files into the disk file named FNAMES.DIR. If FNAMES.DIR already exists, then it is rewritten. The FNAMES.DIR files is used by the FP and FS options. Note that the attribute selection options may also play a part in selecting the files to be logged.

The FP option prints out the user numbers and names of all the files stored in FNAMES.DIR. If FNAMES.DIR is not in the current directory, FP will search along the ZCPR3 path until it finds it or reaches the end of the path.

The FS option scans FNAMES.DIR and the files selected by the user (or implied if no specific file selection option is given) and compares them. If a file exists in FNAMES.DIR but not in the selected files, its name is printed as a missing file. If a file exists on disk but not in the FNAMES.DIR file, then its name is printed as an additional file.

Note that the user should keep in mind what he is scanning for when he uses the file name buffer facility. For instance, if he selects both non-system and system files with the FL option and then defaults to Non-System with the FS option, it is likely that several files will be shown missing even though this may not be true.

The I option (for inspect) allows the user to manually approve each file before its name is placed into FNAMES.DIR.

XIF (version 1.0)

Syntax:
XIF anytext

Function:
If the current Flow State is TRUE, XIF exits all pending IFs. It reduces the IF Level to 0 (no IF in effect).

If the current Flow State is FALSE, XIF does nothing.

Options:
None.

Comments:
None.

Selected Error Messages:
None.

Examples of Use:

```
;=LOOP
XIF
        <statements>
IF 1 3
     GOTO LOOP
FI
```

Z3INS (version 1.0)

Syntax:

Z3INS ufn1 ufn2
UFN1 must be an Environment Descriptor
UFN2 must be a Z3INS Installation File

Function:

Z3INS is the installation program for the ZCPR3 System. All utilities (except ZEX) provided in the ZCPR3 distribution may be installed for a target system by using Z3INS. Z3INS installs the files named in a Z3INS Installation File with data from the Environment Descriptor specified. All files must be ZCPR3 Utilities.

The default file types are ENV for UFN1 (the Environment Descriptor) and INS for UFN2 (the Installation File).

Options:

None.

Comments:

The Environment Descriptor referenced in the command line is a standard ZCPR3 System Environment descriptor which is created by assembling a file like SYSENV.ASM.

A ZCPR3 Installation File is simply a text file containing two types of lines: a comment line, which begins with a semicolon (;), and a line containing an unambiguous file name, which is a file to be installed. An example is shown below:

```
; This is an installation file for my new utilities
util1.com
    util2.com
; UTIL3 is really neat
util3.com
```

Case is not significant. Leading spaces on each line are ignored. Any file name referenced in a command line *must* be unambiguous.

Selected Error Messages:

All error messages are very complete and self-explanatory.

Examples of Use:

```
Z3INS SYS.ENV NEWFILES.INS -- Install the files listed
                               in NEWFILES.INS with the
                               data contained in SYS.ENV
Z3INS NEWENV DIST -- Install the files listed in DIST.INS
                     with the data contained in NEWENV.ENV
```

Z3LOC (version 1.0)

Syntax:
Z3LOC o

Function:
Z3LOC locates and displays the addresses of the running ZCPR3 Command Processor Replacement, BDOS, and BIOS. It may also be run under CP/M to locate and display the addresses of the running CP/M 2.2 Console Command Processor, BDOS, and BIOS.

Z3LOC is also able to display the addresses of a number of ZCPR3 System Segments and data areas if the Z option is given. The Z option should not be given if running Z3LOC under CP/M 2.2.

Options:

Z	Display addresses and data on ZCPR3 System segments and data areas

Comments:
If the Z option is given (Z should only be given if running Z3LOC under ZCPR3), the following additional address information is provided:

o External Path
o Input/Output Package
o Named Directory Buffer
o Shell Stack
o External FCB
o External Stack
o Resident Command Package
o Flow Command Package
o Command Line Buffer
o Environment Descriptor
o ZCPR3 Message Buffer
o Wheel Byte

Selected Error Messages:
Z3LOC generates no error messages. An invalid option invokes a help screen.

Examples of Use:

```
Z3LOC   -- run Z3LOC for CPR/CCP, BDOS, and BIOS display
                (may be used this way under CP/M 2.2)
Z3LOC Z -- display ZCPR3 data as well
                (may be used this way under ZCPR3 only)
```

ZEX (version 3.0)

Syntax:
ZEX // <-- Print Help
or
ZEX <-- Enter Interactive Mode
or

ZEX filename params <-- Process .ZEX or .SUB file as with SUBMIT

Function:

ZEX is a memory-based command file processor. It behaves somewhat like a combination of SUB and XSUB, and because its input source is memory-resident, the execution speed of ZEX is significantly greater than that of SUB/XSUB.

Options:

None.

Comments:

The interactive mode of ZEX executes like the interactive mode of SUB. The user enters command lines until he is satisfied and then terminates the process by entering an empty line (simply hitting RETURN). ZEX then executes the commands in the sequence entered. No parameter passing is permitted in this mode of operation.

The "ZEX filename params" form is like the corresponding SUB form. ZEX will search along the ZCPR3 external path for a command file of the form filename.ZEX or filename.SUB. If a directory is entered which contains both such files, the file of type ZEX will be executed.

Once ZEX has begun execution, it places a ZEX Monitor just under ZCPR3 and builds the command stream under the monitor. Once complete, the BDOS address in locations 6 and 7 is adjusted so that the ZEX monitor and its command stream will not be overwritten by transient programs, and execution begins. Each time the BIOS Console Input routine is called, ZEX supplies the input character.

As with SUB, a ^C from the console aborts execution of a ZEX command stream. Also, as with SUB, if a command follows ZEX in a Multiple Command Line, ZEX appends this command to the command stream.

Unlike SUB, ZEX does not permit nesting of command files. ZEX will simply abort if a ZEX command is encountered in the command stream it is processing.

Unlike SUB, ZEX supports many more embedded commands. Combining the facilities of SUB and XSUB in this case, the embedded commands of ZEX reflect the XSUB-like capabilities of ZEX as well as some new ideas.

These extended control commands are discussed in ZEX Directives, below.

Selected Error Messages:

None discussed.

Examples of Use:

```
ZEX  --  the user now enters a group of commands

ZEX mycmds p1 p2
     --  processing of the file 'mycmds.ZEX', or, if not
         found, 'mycmds.SUB' is performed; 'p1' is
         substituted for $1 and 'p2' for $2
```

ZEX Directives - Control Commands

The ZEX control commands are summarized below. This summary is also displayed via the built-in ZEX help facility.

Cmd	*Meaning*	*Cmd*	*Meaning*
|	insert <CR>	^|	insert <CRLF>
^:	rerun command stream	^.	toggle print suppress
^#	toggle ZEX messages	^$	define default params
^?	wait for user input	^/	ring bell and ^?
^*	ring bell	^"	accept user input
^<	display chars only	^>	stop display
;;	ZEX comment	$n	1<=n<=9 for param
$$	$	$^	^
$|	|	^c	control char

The following commands simply insert characters into the ZEX command stream and will not be discussed in any greater detail.

|	inserts a <CR>	^|	inserts a <CR> <LF> pair
$$	inserts a single $	$^	inserts a single ^
$|	inserts a single |	^c	inserts a control character

The ^* command causes ZEX to ring the bell. It does not insert a BELL character into the command file, as a ^G sequence would. It simply rings the bell and continues processing.

The ;; command introduces a ZEX comment. It and all characters following it up to and including the following <LF> are simply treated as a comment in the ZEX Command File and ignored. Unlike a conventional ZCPR3 comment, the ZEX comment does not take up space in the command stream and does not appear when the command stream is executed.

The ^< and ^> commands are used to bracket characters which are simply echoed by the ZEX monitor and not passed back to the calling program. This causes the characters between these commands to be echoed to the user during execution but not processed by any program. This feature is very useful for embedding comments to be printed at execution time into the command stream. Unlike the ZCPR3 comment form, which is a line beginning with a semicolon, comments enclosed by ^< and ^> may appear anywhere, such as within an editor session.

The ^# command toggles suppression of informative messages generated by ZEX.

The ^. command causes console output to cease until the next ^. is encountered. Character input from the ZEX Monitor continues, but the user does not see it on the screen.

The ^: command causes the ZEX monitor to restart execution of the loaded command stream. The entire command stream, as initially processed by ZEX, is executed again from the beginning.

$n, where 1<=n<=9, will cause the corresponding specified or default parameter to be substituted from the command line.

The ^$ command defines or redefines the set of input command parameters. The rest of the line following the ^$ is treated as a set of parameters separated by blanks.

The ^? and ^/ commands replace the /A and /AB options of SUB. ^? causes ZEX to stop processing and wait for the user to strike either the space bar or the RETURN key before continuing. The user can examine the display at leisure and, if he does not wish to continue, a ^C will abort the command stream. The ^/ command is like ^?, but it periodically rings the bell at the console, summoning the user at critical points in the processing.

Finally, the ^" command causes ZEX to stop providing input from the command stream; the user can then enter whatever he wishes until a special character is output, at which time ZEX will resume providing input. In this case, ZEX can be intimately linked with ZCPR3, and it is intended that the special character that ZEX is waiting for be associated with the ZCPR3 prompt. In my system, I defined the ZCPR3 prompt as a ">" character with the most significant bit set. This is unique and appears only when the prompt comes up on my system.

ZEX Examples:

The following examples illustrate applications employing ZEX. Comments appear out to the side, prefixed by <--.

```
B1>zex
ZEX, Version 1.3
1: ^$ this is fun                   <-- Define 3 params
2: echo $1 $2 $3
3: ^$ hello from happy acres        <-- Define 4 params
4: echo $1 $2 $3 $4
5:
(ZEX Active)                        <-- ZEX is running now
B1>echo this is fun

THIS IS FUN
B1>echo hello from happy acres

HELLO FROM HAPPY ACRES
B1>
(ZEX Completed)
By Your Command >
B1>ed demo.zex                      <-- Demo Command File

NEW FILE
```

```
     : *i
    1:  ed demo.txt                <-- Edit DEMO.TXT
    2:  i                          <-- Insert text while in ED
    3:  This is a test
    4:  This is only a test
    5:  This is a demo of ZEX Control
    6:  ^Z                         <-- ^Z is 2 chars, xlated into
    7:  b0lt                       <-- Ctrl-Z by ZEX
    8:  1l
    9:  0lt
   10:  i                          <-- Input More Text
   11:  ^"                         <-- Allow user to input text
   12:  type demo.txt              <-- When ZEX continues, this
   13:  era demo.txt               <-- is what it does next
   14:
     : *e

B1>zex demo                        <-- Run the command file
ZEX, Version 1.3
(ZEX Active)
B1>ed demo.txt

NEW FILE
    : *i                          <-- ZEX is typing this in
   1:  This is a test
   2:  This is only a test
   3:  This is a demo of ZEX Control
   4:
    :  *b0lt
   1:  This is a test
   1:  *1l
   2:  *0lt
   2:  This is only a test
   2:  *i                         <-- Now user input begins
   2:  I am now typing this line of my own volition   <-- User
   3:  ZEX will allow me to continue doing this until
   4:  it sees the ZCPR3 prompt
   5:                             <-- User types Ctrl-Z
   5: *e                          <-- User types "e"
```

```
(ZEX Active)

B1>type demo.txt                <-- ZEX resumes
This is a test
I am now typing this line of my own volition
ZEX will allow me to continue doing this until
it sees the ZCPR3 prompt
This is only a test
This is a demo of ZEX Control


B1>era demo.txt
DEMO .TXT
B1>
(ZEX Completed)
By Your Command >

B1>ed demo.mac                   <-- Now to use ZEX for
                                 <--   program assembly
NEW FILE
    : *i                         <-- User types program
   1:        ext     print
   2:
   3:        call    print
   4:        db      'Hello, World ... It''s Another Day',0
   5:        ret
   6:
   7:        end
   8:
    : *e


B1>type a:m80.zex                <-- M80.ZEX command file
; M80.SUB -- MACRO-80 Assembler and Linker
M80 =$1
; Please Type $^C if Error(s) Exist - ^?
ERA $1.BAK
ERA $1.COM
L80 /P:100,$1,A:SYSLIB/S,$1/N,/U,/E
ERA $1.REL
```

```
; Assembly Complete

B1>zex m80 demo                    <-- Run command file on pgm
ZEX, Version 1.3
(ZEX Active)
B1>; M80.SUB -- MACRO-80 Assembler and Linker
B1>M80 =DEMO

No Fatal error(s)

(ZEX Active)

B1>; Please Type ^C if Error(s) Exist -  <-- User can abort now
                                         <--   if he wishes
B1>ERA DEMO.BAK
DEMO .BAK
B1>ERA DEMO.COM
No File


B1>L80 /P:100,DEMO,A:SYSLIB/S,DEMO/N,/U,/E

Link-80  3.44  09-Dec-81  Copyright (c) 1981 Microsoft
Data    0100    01C5     < 197>

35936 Bytes Free

Data    0100    01C5     < 197>

35936 Bytes Free
[0000   01C5         1]

(ZEX Active)

B1>ERA DEMO.REL
DEMO    .REL
B1>;  Assembly Complete
B1>
(ZEX Completed)
By Your Command>demo                <-- Run pgm now
```

```
Hello, World ... It's Another Day
B1>
```

4 On-Line HELP Subsystem

Overview of the HELP Command

The HELP Command provides interactive, online assistance in using ZCPR3 in general and specific ZCPR3 commands in particular.

HELP pulls in files named <FILENAME>.HLP from disk and displays these to the user in a paged mode. These files are of two basic types: indexed and non-indexed.

Indexed HELP files, of which HELPSYS.HLP is an example, start with an index. When HELP loads an indexed file, it displays this index to the user and allows him to select as many entries as he desires, in any order, by simply typing the letter(s) corresponding to his selection. Once the user has made his selection, HELP will look up the associated body of text and display it to him in a paged mode. When the user has finished reading, HELP returns him to the index menu. Typing a Control-C will return the user to ZCPR3.

There are two types of indexed HELP files: user-indexed and HELP-indexed. A *user-indexed* file (of which HELPSYS.HLP is an example) is one in which the writer of the file is allowed to create the image of the index on his screen in the form which will be displayed to the user of the HELP file.

A *HELP-indexed* HELP file is one which contains a list of the options at the beginning of it; HELP automatically creates the menu, assigning sequential letters (A, B, etc) to the menu options.

Non-indexed files do not start with an index. In such cases, HELP will immediately display the contents of the file to the user and, when the user has finished looking at it, will return to ZCPR3.

HELP is menu-driven, and all the commands available to the user at any given time are displayed to him.

The version of HELP described in this manual is designed to work with the ZCPR3 system and take advantage of some of its special features.

How to Use the HELP Command

The HELP Command is executed in one of three ways:

1. By just typing 'HELP'
2. By typing 'HELP FILENAME', where FILENAME is the name of a disk file named FILENAME.HLP
3. By typing 'HELP FILENAME.TYP', where FILENAME.TYP is the name of a file created in the format of a help file

If the user types just 'HELP', he will review the file HELP.HLP, which should contain a brief summary of how to use the HELP command. For all other forms of the HELP command, the user will see the specified help file information. Generally speaking, the name of the help file should be indicative of its subject–i.e., CPM.HLP should contain help information on CP/M.

HELP File Search Hierarchy

Whenever HELP looks for a specified HELP File (either from the HELP Command or from an information section which specifies a Node [see later]), HELP will perform a search for the indicated file. This search goes as follows:

1. Under ZCPR3, HELP will follow the command-search path, searching the current directory (disk and user) first.
2. If the HLP file is not found in the current directory, HELP will search along the ZCPR3 path for it.
3. If the HLP file is not found along the ZCPR3 path, then HELP will look in the directory named "HELP" for the indicated file. This is a major difference between HELP and other ZCPR3 utilities.
4. If the HLP file is not found, HELP will print an error message.

Moving Around within the HELP Command

Once the user is running HELP, he is given a set of commands by which he can display the particular items of information he is interested in.

After issuing the HELP command, the user will come up in one of two modes (depending on the type of HELP file referenced). In indexed mode, a menu of topics is displayed to the user and he can select the desired topic by typing the character in front of the topic title. In non-indexed mode, no menu is displayed; instead, the entire file is viewed as one "information section."

An *information section* is a collection of screen displays (one screen full of text) called *frames*. Typically, an information section should contain a logical grouping of related data on a particular topic. In indexed mode, each menu topic refers to an information section. By selecting a topic, the user is placed into the corresponding information section. In non-indexed mode, the entire HELP file is one information section.

Moving From the Menu

At the menu of a HELP file, the user has two basic choices: to select a menu topic for review, or to exit to ZCPR3. If a menu topic is selected, the user is placed into the corresponding information section.

A third choice is sometimes available at the menu level: to move up to the previous HELP Level. Some information sections are entire HELP files in their own right, which can be accessed independently of the HELP file the user is currently in. If the user enters one of these information sections, the name of the current HELP file is saved and the new HELP file is loaded. When this happens, the user is placed at the next HELP level.

HELP levels start at 0 and increase each time the user calls a new HELP file from his current level. Thus, he starts at level 0, and the first HELP file he calls puts him at level 1. If he now calls another HELP file, this puts him at level 2. From level 2 he may have the option of exiting, either to ZCPR3 (which would return him to help level 0) or to the previous level (1).

The HELP files are organized in a tree data structure. To get to a particular HELP file, the user starts at the root of the tree and then climbs up and down the trunk and branches to various levels, or nodes. From each node, the user may only move up or down the tree—he can't cross over to a node at the same level without first moving down the tree and then back up. To illustrate, consider the following:

```
 Node A      Node B                                 HELP Level
--------   ----------                                    5
 \ Node C   /
--------------                                           4
       \ Node D        Node E
    ------------     ----------------                    3
           \      Node F      /    Node G
           --------------------   --------               2
                     \       Node H    /    Node I
                     -------------------   --------      1
                               \Node J       /
Root of Tree -->             -------------------         0
```

In the above example, the user must always start at the root of the tree (Node J). This is analogous to HELP Level 0, which is where the user is placed when he issues the HELP Command. To get to Node C, for example, the user has to climb the tree from Node J to Node H to Node F to Node D to Node C. This would be like the user entering four node-type information sections, in which different HELP files are successively loaded.

Now that the user is at Node C, let's say that he wants to go to Node E. Under the HELP System, there are two ways to do this:

1. Jump to the ground and then climb back up to Node E. Here, the user would jump from Node C to Node J and then go to Node H to Node F to Node E. Under HELP, the user can do this by exiting to ZCPR3 and then reissuing the HELP Command or by issuing the Root Command (.); once at the root of the tree, he then climbs it again by entering the appropriate information sections.
2. Climb down the tree and then back up. The user would move from Node C to Node D to Node F and then back up to Node E. The HELP user can go to the previous level by issuing the Up Level (^) command. In this example, he would issue the Up Level command twice and then go back down.

Moving Within an Information Section

Once the user is within a textual information section, he has several capabilities for moving within this section or to another information section.

First, to leave an information section, the user can return to the menu (if the current HELP file is indexed) or return to ZCPR3. Additionally, if the user is not on the root (HELP Level 0), he can return to the previous HELP Level (Up Level). If the user is not in an Indexed HELP file, moving forward beyond the End of Information (EOI) will return him to ZCPR3 if he is at HELP Level 0, or to the previous HELP Level if not.

The data within an information section is arranged sequentially. Consequently, the user can move forward to the next frame, or backward to the previous frame or to the beginning of the information section. The user cannot move backward beyond the beginning of the information section; if he tries to do so, a bell is sounded. Also, if the user tries to move forward beyond the End of Information (EOI), he is returned to the menu, returned to ZCPR3, or returned to the previous HELP Level as described above.

HELP Status and Command Prompts

Whenever the HELP system is in use and an information section is being displayed, the bottom line of the screen displays some status information and the prompts for HELP commands available to the user.

The status indicators appear at the extreme left of the bottom line; they are followed by the command prompts. The status indicators may take the following forms:

```
<Nothing> .....command prompts....
     \__The user is at the menu of Level 0

fff: .......command prompts....
 \__Current Frame Number within information section
      (the user is at Level 0)

Level 111/ .......command prompts....
      \_Current Level Number (The user is at a menu frame)
     (this is displayed only if the user is NOT at Level 0)

Level 111/fff: ......command prompts....
      \   \__Current Frame Number within information section
       \_Current Level Number
      (this is displayed only if the user is NOT at Level 0)
```

The command prompts take one of the three forms shown below, depending on the HELP files in use:

```
^C=ZCPR3 ^=Level .=Root M=Menu S=Start L=Last P=Print -
   \     \    \     \    \    \    \__Print Info/Frame
    \     \    \     \    \    \__Goto Last (Previous) Frame
     \     \    \     \    \__Goto Start of Info Section
      \     \    \     \__Goto Menu of HELP File
       \     \    Root if NOT at Level 0
        \     \(this is displayed only if NOT at Level 0)
         \     \__Goto Previous Level
          \(this is displayed only if NOT at Level 0)
           \__Return to ZCPR3
```

```
EOI ^C=ZCPR3 ^=Level .=Root M=Menu S=Start L=Last P=Print -
\   \_______\________\_______\________\_______\______\_Same
 \                                                    as Above
  \_User is at the End of Information (end of information
                                                    section)

Type ^C=ZCPR3 ^=Level .=Root or Enter Selection -
      \       \       \           \_Enter letter of desired
       \       \       \            information section
        \       \       \_Goto Root
         \       \ (this is displayed only if NOT at Level 0)
          \       \_Goto Previous Level
           \       (this is displayed only if NOT at Level 0)
            \_Return to ZCPR3
```

Summary of User Commands

Cmd	*Meaning*
^	Go to Previous Level
.	Go to Root Level
M	Go to Menu of Current HELP File
S	Go to Start of information section
L	Go to Previous Frame
CR	(Carriage Return or Space) Go to Next Frame
^C	(Control-C) Return to ZCPR3
P	Print Current Screen Display (Frame) or information section

Printing HELP Files

The printing of HELP files can be done in two ways: by using the HELPPR Utility of ZCPR3, or by using the Print function contained within the HELP Utility itself.

The HELPPR Utility prints an entire Help File. It acts a lot like the PRINT command, and it has a variety of options, including the ability to plan for printer output and to support paging and other "appearance-enhancing" features.

The Print Function within HELP is used for quick printouts. It does not page or perform anything more exotic than simply printing out what is contained in a part of a Help File. When the P option is given, the current screen is printed immediately. The user may also issue a ^P command (not displayed on any menu), in which case the entire current information section is printed.

This Print Function is provided as a convenience to the user. It allows the user to review the Help File, and, when he sees a particular screen display or information section which he values enough to want to have around for future reference in hardcopy form, he can simply tell HELP to print it. This capability is intended to support the concept of establishing HLP files as a convenient and flexible way to pass documentation of programs to the user on disk, while also making it easy for him to print it out if it is of significant interest to him.

For instance, a HLP file which refers to a new program may include an information section or one frame which contains a command summary. The user can simply print this without having to print the entire HLP file.

As another example, the HELP subsystem may be used by a homemaker to store her recipes. If these are organized, using the tree structure, into reasonable categories (such as roasts, desserts, etc), while reviewing the recipes she may find one she wishes to try for the evening's meal or to pass on to a friend. If the recipe covers only one screen, a Frame Print is very convenient.

HELP Error Messages

The following are the error messages issued by HELP, and their meanings:

File not Found The specified HELP File cannot be found.

AFN Not Allowed The specified HELP File is ambiguous (contains the character "*" or "?"). This is not allowed.

<BELL> The user issued an invalid command.

EOF on HELP File In searching for an information section, HELP ran into the end of the HELP File. The Indexed HELP file is improperly structured (more index entries than information sections).

Node Level Limit The limit of the nesting of the HELP Levels is exceeded. HELP limits the number of HELP Levels that can be traversed to 10 (default, which can be changed), and an attempt was made to enter HELP Level 11 (or default + 1).

Mem Full The selected HELP File is too large to load into the available memory in the user's computer system. The HELP File should be reduced in size; using HELP Levels (Node references) in the information sections is a good way to do this.

How to Write HELP Files

Files used by the HELP program are either simple CP/M-standard files of ASCII text or ASCII files generated by the WordStar text editor/formatter. These files are of two basic types: indexed or non-indexed; both types have the same basic format.

Grouping of Information. Information displayed to the user is grouped by the index in indexed HELP files and may also be grouped by lines beginning with form-feed (^L) characters. Grouping is an effective way to organize information logically so that meaning will be more clear to the user and units of information will not overrun screen display boundaries.

The information displayed to the user is organized into logical units called *information sections*, and screen-sized displays called *frames*. Using a text editor, the user can create his own HELP files and organize his information as he desires for display by the HELP subsystem.

Non-Indexed HELP Files. Non-indexed HELP files are identified by having a colon (:) as the first character of the first line in the file. The file consists of ASCII text lines, each line being terminated by a carriage return (0DH) followed by a line feed (0AH). The information section in such a help file consists of the text following the leading colon; this text may be terminated either by a new line that starts with a colon (thus beginning a new information section), or by the end-of-file marker (control-Z, 1AH).

Indexed HELP Files. Indexed HELP files are simple ASCII files which do not start with a colon (:) as the first character of the file. An indexed HELP file may be HELP-indexed or user-indexed.

A HELP-indexed HELP file is identified by the fact that the first character of the first line is alphanumeric (not a punctuation mark). The index entries are contained in one or more such ordinary ASCII text lines; HELP labels these lines with alphabetic identifiers (A, B, C, etc) during the display of the index. The index is followed by ASCII lines comprising one or more information sections. The first line of each *information section* in the file *must* have a colon in column 1.

A user-indexed HELP file is identified by having a semicolon (;) as the first character in the file; the semicolon *must* be immediately followed by a CR/LF sequence. The text that starts on line 2 is displayed literally to the user as the menu. Since the displays generated by HELP are screen-oriented, the menu may be longer than 24 lines provided that there is a properly installed TCAP entry in the Environment Descriptor. The menu is followed by one or more information sections. The start of each information section is denoted by a line starting with a colon (:) and followed by a series of characters (spaces are not significant between them) which are the index letters. When the user runs HELP on this file and types a selection letter, HELP searches through the file, looking for an information section whose first line contains the character typed by the user. If the character was a letter, it is automatically capitalized by HELP (in both the user input and the information section lines).

Note that a colon is not a valid option letter, since this character has a special meaning to HELP. If a colon is encountered after column 1 in an information section heading line, the scan for option characters stops and subsequent characters may be interpreted as another help file name to be invoked by option letters already found in the line.

Tree Structures

The Indexed HELP File is divided into information sections, each of which starts with a colon (:). There are two basic types of information section:

1. Information sections containing only textual material. This type of information section may appear in both HELP-indexed and user-indexed files; it begins with a single colon and contains reading material which is organized into frames, each of which is equal to one screen display.

2. Information Sections which reference other HELP files. This type of information section appears *only* in HELP-indexed files; it begins with two colons (::) instead of one. The two colons are immediately followed by the name of the HELP file (the HELP file type may be optionally specified).

In user-indexed HELP files, this type of information section contains the index characters followed by a colon (:) and the name of the HELP file (the file type is optional).

HELP-Indexed File Skeletons

Skeleton outlines of HELP-indexed files of both types are shown below as samples to follow.

Example 1: Text information sections

```
INDEX ENTRY
INDEX ENTRY
```

```
    :Title for Type 1 information section
        <text>
    :Title for Type 1 information section
        <text>
    EOF marker
```

Example 2: Node information sections

```
    INDEX ENTRY
    INDEX ENTRY
    ::HLPFILE                    <-- for HLPFILE.HLP
    : [next information section]
    ::HLPFILE.TYP                <-- for HLPFILE.TYP
    : [next information section]
    EOF marker
```

User-Indexed File Skeleton

```
;
-------------------
      [ Menu lines]
-------------------
:x
      [ Information Displayed for Selection X ]
:a
      [ Information Displayed for Selection A ]
:1 b
      [ Information Displayed for Selections 1 or B ]
:f :HELPFILE    [ HELPFILE.HLP is invoked by Selection F ]
: z
      [ Information Displayed for Selection Z ]
EOF marker
```

Accessing Video Attributes

The displays generated by HELP are screen-oriented. Under ZCPR3 with a properly installed TCAP entry in the Environment Descriptor, HELP is able to highlight information on the screen and create "flashy" displays by using the clear screen command appropriate to the user's terminal. Use of this feature is automatic; each frame is preceded by a clear screen.

The writer of a HELP file can turn text highlighting on and off anywhere in the HELP file by embedding the following commands into the text:

```
    ^A (binary 1) -- turn highlighting ON
    ^B (binary 2) -- turn highlighting OFF
```

It is recommended that if highlighting is turned on, then it should be turned off in the same line. Example:

```
^Athis is highlighted^B while this is not
```

The file HELPSYS.HLP is an example of one which extensively uses highlighting. It is also a user-indexed HELP file.

Tree Structure of HELP

The diagram below illustrates how tree structures can be implemented under HELP. A new node of the tree is created whenever an information section references a HELP file instead of merely containing text. Each node becomes the base of a new tree, which itself may contain references to other HELP files.

From the diagram, we see that SubHelp Level 3 contains two HELP files. These can be entered from Information Section 2 and Information Section 3 of SubHelp Level 2. By simply entering one of these two information sections, the appropriate HELP file is loaded and the user is placed at the next level. From these HELP files, the user may move within the HELP file itself or move up to the previous level (naturally, the user always has the option to exit to ZCPR3).

```
                     --  Basic HELP File --
|Info Sect 1 |Info Sect 2 |Info Sect 3 |Info Sect 4 |     L0
| Text       | HELP File  | Text       | HELP File  |
                 /      \                   /    \
      --  SubHelp File 1 --          -- SubHelp File 2 --  L1
|Info Sect 1 |Info Sect 2 |     |Info Sect 1 |Info Sect2|
| Text       | HELP File  |     | Text       | Text     |
                 /    \
     --  SubSubHelp File 1 --                              L2
|Info Sect 1 |Info Sect 2 |Info Sect 3 |
| Text       | HELP File  | HELP File  |
                 /    \       /    \
-- Sub3Help File 1 --    -- Sub3Help File 2 --             L3
|Info Sect |            |Info Sect 1 |Info Sect 2|
| Text     |            | Text       | HELP File |
                                         /    \
                               --  Sub4Help File --        L4
                             |Info Sect 1 |Info Sect 2|
                             | Text       | Text      |
```

Sample HELP Files with Trees

The following listings show the source to three HELP Files. DEMO.HLP provides the Root Node to a tree which includes DEMO2.HLP and DEMO3.HLP as subnodes. Additionally, DEMO3.HLP has a subnode which references DEMO.HLP, so we have a recursive tree structure.

DEMO.HLP

```
TEST 1 - OK                  <-- Menu
TEST 2 - SIMPLE NEST
TEST 3 - INVOLVED NEST
TEST 4 - OK
:TEST 1                      <-- First Info Section (Text)
THIS
IS
TEST
1
::DEMO2                      <-- 2nd Info Section (Node)
::DEMO3                      <-- 3rd Info Section (Node)
:TEST 4                      <-- 4th Info Section (Text)
TO BE, OR NOT TO BE, THAT IS THE QUESTION!
  ...
TO TAKE ARMS AGAINST A SEA OF TROUBLES AND BY OPPOSING END THEM.
TO DIE, TO SLEEP ... TO SLEEP, PERCHANCE TO DREAM.
AYE, THERE'S THE RUB! FOR IN THAT SLEEP, WHAT DREAMS MAY COME!
```

DEMO2.HLP

```
  :TEST 2                      <-- No Menu -- 1 Info Section
  THIS
  IS
  TEST
  2
```

DEMO3.HLP

```
  TEST 3A                      <-- Menu
  TEST 3B
  TEST 3C
  :TEST 3A                     <-- First Info Section (Text)
     THIS
     IS
     TEST
     3A
     the rain in Spain falls mainly in the plain
```

```
:TEST 3B                        <-- 2nd Info Section (Text)
   THIS IS TEST 3B
::demo                          <-- 3rd Info Section (Node)
```

5 Menu Subsystem

Overview

The menu subsystem provides two menu-oriented command preprocessors, MENU and VMENU. Each of these draws its menu data from an associated file designated *.MNU (for MENU) or *.VMN (for VMENU). For those users who wish to construct their own menus, two diagnostic tools are available to check the syntax of a new data file: MENUCK validates *.MNU files, and VMENUCK validates *.VMN files.

The difference between the two preprocessors is that MENU is basically line-oriented, and can be used whether or not TCAP data is included in the environment descriptor, whereas VMENU is screen-oriented, relies heavily on screen characteristics defined in TCAP, and can generate more elaborate and "flashy" displays. Generally speaking, anything that can be done with MENU can also be done with VMENU, but the reverse is not true.

Given the similarities between the two preprocessors, this chapter describes the menu subsystem in terms of the capabilities of VMENU, pointing out the differences between MENU and VMENU where these are significant. Thus, all references to VMENU also apply to MENU except where otherwise stated. When discussing areas that are identical for both programs, we shall refer to "(V)MENU" as a shorthand notation for "VMENU and/or MENU" (or some similarly cumbersome inclusive expression). VMENU works with menu files of type *.VMN; MENU works with files of type *.MNU. This chapter will refer to all menu files with the VMN file type, and the reader should be aware that if he is dealing with MENU, the file type is MNU.

Preprocessor Operation

(V)MENU is a ZCPR3 Menu-Oriented Command Preprocessor. It acts as a front-end to ZCPR3, providing a menu-oriented user interface to ZCPR3. Its function can be represented by the following diagram:

```
                  -------     -------     -------     -------
                  |V     |    |Z     |    |B     |    |P     |
User              | M    |    | C    |    | D    |    | g    |
Menu         ---->|  E   |--->|  P   |--->|  O   |--->|  m   |
Command           |   N  |    |   R  |    |   S  |    |   s  |
                  |    U |    |    3 |    |      |    |      |
                  -------     -------     -------     -------
```

The "User Menu Command" is a single character that the user strikes which instructs (V)MENU to perform a function. Once (V)MENU begins processing this function, it builds a command line for ZCPR3, optionally asking the user for further input (such as a file name), and then passes the command line to ZCPR3 via the Command Line Buffer. ZCPR3 then runs the command line and returns to (V)MENU.

(V)MENU builds command lines based on simple input from the user. The user need never know what the actual command line is. The command line itself is always of the form:

```
<command> <optional user input>
```

As an example, a command built by (V)MENU to run XDIR with user input for a file name specification, could look like the following:

```
XDIR <user input>
```

When (V)MENU is executed, it looks for the file MENU.VMN in the current directory. If it finds one, it loads it and begins processing. If it does not find one, it simply exits. A file name may be specified in the (V)MENU command line to select a file other than MENU.VMN.

The MENU.VMN file can contain up to 255 menus to be processed by (V)MENU. The (V)MENU will begin processing at the first menu in MENU.VMN.

(V)MENU itself is a COM file, like any other program under ZCPR3. Unlike most other programs, however, it generates command lines to be executed by ZCPR3 and stores its return command in the Shell Stack. In this way a loop is set up:

```
-->--+->-  (V)MENU ->- ZCPR3 ->-+
     ^                          v
     |                          |
     +-<-  Command Line   -<----+
```

Only (V)MENU itself or a ZCPR3 tool like SHCTRL can terminate this loop. A MENU.VMN file can be set up to execute any ZCPR3 command or sequence of commands. The MENU.VMN file can also be set up to not allow the user to leave (V)MENU, to allow him to leave (V)MENU at will, or to allow him to leave (V)MENU only if he knows a password.

This chapter is divided into two basic parts. The sections on "Using (V)MENU" and "Summary of (V)MENU Commands" are designed to be read by a person wanting to use (V)MENU but not wanting to learn how (V)MENU works or how to program it. These sections describe how to move from one menu to another, how to issue (V)MENU commands, and how to leave (V)MENU if the option is presented to him.

The other sections describe the programming aspects of (V)MENU and are intended to be used as a reference for the (V)MENU programmer. The (V)MENU command programming summary is especially useful because it provides a summary of the commands which the (V)MENU programmer may issue to (V)MENU within a *.VMN file.

Using MENU and VMENU

MENU Invocation. When MENU is first invoked, one of three things will happen:

1 A menu will appear and be paged up to fill the screen; a command prompt will appear at the bottom of the menu.
2 A menu will appear and not be paged up to fill the screen; a command prompt will appear at the bottom of the menu.
3 A command prompt will appear with no menu; this is called the Expert mode.

If at any time a menu display is garbled, or you wish to see the current menu (as sometimes happens when you are in Expert mode), just strike the Return key. The Return key refreshes the menu at all times.

There can be up to 255 menus in one MENU.MNU file; the command prompt varies to reflect this. For example, if there were only one menu in the file, and the option to abort to ZCPR3 were not enabled, then the command prompt would take its simplest form, looking like:

```
Command(<CR>=Menu) -
```

At this time the user may strike Return to refresh the display or strike the character corresponding to a menu selection. Striking any other character causes the bell to sound.

For instructions on moving from one menu to another, refer to "Using VMENU," below. One option available under MENU but not under VMENU involves access to a System Menu, intended to give privileged users access to special commands that other users are not allowed to run.

Access to a System Menu (if one is available) is gained by typing the command "$". MENU will respond:

```
Pass?
```

The user is given only one chance to type the correct password for entering the system menu. If he enters an invalid password, the message "Password Error" will be displayed and he will be returned to the menu he came from. If he entered the correct password, the system menu will be displayed. This is always the last menu in the file and its command prompt is:

```
Command (<CR>=Menu, *=1st Menu, <=Previous Menu) -
```

If the user is at the last menu before the system menu, the ">" command will not allow the user to enter the system menu, even though, technically, this is the "next menu." The only way to enter the system menu is via the "$" command and a valid password.

Using VMENU

When VMENU is first invoked, it will be installed as a Shell, and control will return to the ZCPR3 command processor for the next command in the line. When the command line is exhausted, ZCPR3 will realize that a shell has been installed and invoke VMENU as a shell.

VMENU will then come up, load the names of the files in the current disk directory, load the menu file, and display up to sixteen files and the first menu in the menu file to the user. The user will then be prompted for a command.

If at any time a Menu Display is garbled, just strike the ^R key. ^R refreshes the menu at all times.

The prompt which appears at the bottom of the Menu display has the following general form:

```
Command (<CR>=Menu,^C=Z3,*=1st Menu,<=Prev Menu,>=Next Menu) -
```

There can be up to 255 menus in one VMENU.VMN file. The VMENU command prompt varies to reflect this. For instance, if only one Menu is present and the option to exit from VMENU to ZCPR3 is not available, then the Menu Command prompt would take its simplest form, looking like this:

```
Command (<CR>=Menu) -
```

Strike ^R at this time to refresh the Menu Display or strike the character of a Menu Option. These are the only choices, and striking a character which is not the RETURN key or a menu option causes the bell to sound.

If the the option to exit to ZCPR3 is available and there is only one Menu in the MENU.VMN file, then the command line will look like this:

```
Command (<CR>=Menu,^C=Z3) -
```

The option of aborting to ZCPR3 by striking Control-C (hold down on the Control, or CTRL, key and strike the letter C) is now available. This will exit VMENU and return to ZCPR3.

On a brief note on option letters before going on. If one of your options is a letter in the range A-Z, then case is not significant, and you can invoke the option A, for example, by striking either an upper- or a lower-case A.

If there is more than one menu in the *.VMN file, the command line options become slightly more complex, but they are still quite easy to follow.

In the following examples, assume that the option to exit to ZCPR3 is off, so the "^C=Z3" option will NOT appear.

From the first menu in the file, the command line will look like the following:

```
Command (<CR>=Menu,>=Next Menu) -
```

To advance to the next menu, strike the ">" or the "." character. On most keyboards, ">" is the shift of the ".", so VMENU permits easy movement without having to worry about shifting the keyboard all the time.

If the last menu in the file is on the screen, the command line will look like the following:

```
Command (<CR>=Menu,*=1st Menu,<=Prev Menu) -
```

This allows the user to strike the "*" character to jump back to the first menu in his *.VMN file. If "<" or "," is struck ("<" is usually the shift of the ","), then the user will back up one menu to the previous menu in the file.

If the user is somewhere in the middle of the MENU.VMN file, his command line will look like this:

```
Command (<CR>=Menu,*=1st Menu,<=Prev Menu,>=Next Menu) -
```

Again, "*" will go directly to the first menu, "<" or "," will go to the previous menu, and ">" or "." will go to the next menu. Striking the RETURN key will refresh the menu display.

In summary, moving about within VMENU is quite easy. "*" moves the user to the first menu, "<" to the previous menu, ">" to the next menu.

Summary of MENU and VMENU Commands

The full menu command line looks like the following:

```
Command (<CR>=Menu,^C=Z3,*=1st Menu,<=Prev Menu,>=Last Menu) -
```

The available commands are:

Command	*Function*
<CR>	Refresh menu display (MENU only)
^R	Refresh menu display (RETURN Key) (VMENU only)
^C	Exit to ZCPR3 (Control-C)
*	Jump to the first menu
< or ,	Jump to the previous menu
> or .	Jump to the next menu
$	Enter system menu (MENU only)
other	Menu Option or Invalid Command; letters are automatically capitalized, so a=A

Programming *.MNU and *.VMN Files

Data files *.MNU (used only with MENU) and *.VMN (used only with VMENU) have identical structures, but MENU has a number of options not available in VMENU.

The *.MNU (or .VMN) file is simply a text file which may be created with any conventional CP/M editor, including WordStar. (V)MENU ignores the most significant bit of all bytes, so editors such as WordStar, which occasionally set this bit, can be used.

All *.MNU (or .VMN) files have the same general structure. The first line is either a global option line or the beginning of a menu display. If a global option line, it begins with the character "-", and this character is immediately followed by global option characters. The global option line, then looks like this:

```
-option
```

After the global option line, if any, comes the first menu. Each menu is structured as follows:

```
#option
      <Text of Menu Display>
#
menu commands
```

The following are two sample Menu File structures:

```
-option                       #option
#option                            <Text>
     <Text>                   #
#                             commands
commands                      ##
#option
     <Text>
#
commands
```

```
##
```

Options. VMENU has only one option character—"X"—which tells VMENU to allow the user to exit to ZCPR3. In using it, case is not significant. The X option enables the facility which permits the user to type ^C and return to ZCPR3.

MENU has three additional options, not available to VMENU:

C	Display command line to user
D	Display menu to user
P	Page Out menu display

The C option displays the command line built by MENU to the user. This option is primarily intended for debugging purposes; however, it can also be instructive to the user. The D option displays the menu to the user; if the display function is not turned on, we are in the Expert mode and the commands are available without a menu display. The display can be turned on at any time by striking Return. The P option clears the screen by issuing 24 successive <CR-LF> sequences to scroll the menu up off the screen. It is useful for keeping only the current commands and options on the screen, but annoying to a user accessing the system via a 300-baud modem; the menu programmer therefore is given a means of turning off this scrolling.

When (V)MENU first comes up, all options are turned off. The user cannot exit to ZCPR3. The global options line which, if present, is the first line of the file, turns on the specified options for the course of the session in general. That is, if a global options line like

```
-DPx
```

is used (case is insignificant), then menu display, paging, and exit to ZCPR3 are enabled for all menus. However, if some of the menus in the file ought to deny exit to ZCPR3, then the -x option may be temporarily turned off for those menus. This is done by presenting the X option on the first line of each such menu immediately after the "#" character. If the X option is NOT included in the global options line, it is turned OFF for all menus in general. The default selected by using the global options line is overridden on a per-menu basis by the local menu options.

Example:

```
-x
#x
      No exit to ZCPR3 is permitted
#
commands
#x
      No exit to ZCPR3.
#
commands
#
      The user may exit to ZCPR3.
```

```
#
commands
##
```

***.MNU and *.VMN Commands**

This section describes the technique and options available for creating command lines in menu files. The information herein is organized into the following subject areas:

o Syntax of the command line
o :nn Option
o ! Option
o "text" prompts and input
o Variables ($D, $U, $Fn, $Nn, $Tn, $Pp, $$)
o Highlighting (^A, ^B)

Command Structure

The commands in a menu file follow a simple structure. Each command occupies only one line, and blank lines in the command group are not permitted. The command line is structured as follows:

```
    l[o][command]
```

where:

l is the single character used to invoke the command;
note that it may be upper- or lower-case.
o is an opening option, which is one of:
:nn -- go to Menu nn
! -- have (V)MENU wait when the command is finished
command is an optional ZCPR3 command; note that if the option is ":nn",
then a command here makes no sense.

:nn Option

The ":nn" option tells (V)MENU to move to a different menu in the *.VMN file. The first menu is number 1. Example:

```
-x
 #
        1st Menu: A - Goto Menu 2 3 - Goto Menu 3
 #
 a:2
 3:3
 #
             2nd Menu Command: 3 - Goto Menu 3
 #
 3:3
 #x
             3rd Menu Command: 2 - Goto Menu 2
```

```
#
2:2
##
```

In the first menu, the user may strike:
  "a" or "A" to goto Menu 2
  "3" to goto Menu 3
  ">" or "." to goto the next menu (Menu 2)
  ^C to goto ZCPR3
In the second menu, the user may strike:
  "3" to goto Menu 3
  "*" or "<" or "," to goto Menu 1
  ">" or "." to goto Menu 3
  ^C to goto ZCPR3
In the third menu, the user may strike:
  "2" or "<" or "," to goto Menu 2
  "*" to goto Menu 1

! Option
The "!" option causes (V)MENU to pause after the command line has been processed and ask the user to "Strike Any Key" before continuing. In this way, if a command generates information to be read by the user before (V)MENU clears his screen, the "!" option may be used to give the user all the time he wants to read this display.
" Prompt
Embedded within any command line may be a prompt for user input. This prompt takes the form of
    "prompt to user"
When encountered, (V)MENU will advance to the next line and print the text contained within the quotes. (V)MENU will then wait for the user to enter any text he desires followed by a RETURN. At this point, the text entered by the user is capitalized and placed into the command line at the point of the prompt.
If the prompt appears at the end of a (V)MENU command line, the trailing quote is not required. As many prompts as desired may appear within a (V)MENU command line. Examples:

```
-x
#
        A - Run XDIR without Pause or Input
        B - Run XDIR and Pause before Returning to (V)MENU
        C - Run XDIR, Allow User Input, and Pause before
             Returning to (V)MENU
        M - Run MCOPY, Allow User Input of Dest Dir, Allow
             User Input of Source Dir and File, and Pause
             before returning to (V)MENU
        Z - Run Any ZCPR3 Command and Pause before
```

```
             Returning to (V)MENU
#
m!mcopy "Destination Dir? "="Source DIR:AFN? "
z!"Enter Command Line --
      axdir
      b!xdir
      c!xdir "Enter Ambiguous File Name --
      ##
```

Note the space right before the "Prompt form in the C command. This space is significant to keep the command and user input from running together. This "run together" is desired for the Z command. Also note the dual prompt for the M command.

With the M command, the following prompts will appear (and sample input):

```
Destination Dir? BACKUP:
Source DIR:AFN? *.TXT
```

and the following command line is built:

```
MCOPY BACKUP:=*.TXT
```

The command text specified in the (V)MENU command line can contain embedded variables which (V)MENU will expand when the command line is processed. These variables, which are denoted by a dollar sign ($) followed by one or two characters, are defined as follows:

Variable	Expands as
$D	Current Disk
$U	Current User Area
$Fn	FILENAME.TYP for ZCPR3 System File n
$Nn	FILENAME for ZCPR3 System File n
$Tn	TYP for ZCPR3 System File n
$Pp	Name of File being Pointed to (VMENU only)
$$	Place a single $ in command line

These variables can also be used in the menu display itself, and their values will be substituted when the display is generated. Example:

```
-x
#
              Menu to Run M80 Assembler
                              Current File: $F1
                              Directory: $D$U
       F - Define File
       E - Edit $F1           P - Page $F1
       A - Assemble $F1
```

```
#
fsetfile 1 "Filename? "
eedit $fl
azex m80 $nl
ppage $fl
##
```

Notes:

1. The ZCPR3 utility SETFILE is used to define the name of a ZCPR3 System File. Four System Files are available, and they can be referenced by $F1 to $F4, $N1 to $N4, and $T1 to $T4.
2. The A command shows the execution of ZEX. (V)MENU is a true ZCPR3 Shell, and therefore ZEX commands can be issued from it and will run on top of it. The prompt for the ZEX command lines will be "Menu>".
3. Assuming that "MYFILE.MAC" is assigned to the ZCPR3 System File 1 and the user is logged into disk B user 1, the following screen shows how the display and the resulting command lines will be expanded when execution occurs:

Display

```
           Menu to Run M80 Assembler
                          Current File: MYFILE.MAC
                          Directory: B1
      F - Define File
      E - Edit MYFILE.MAC          P - Page MYFILE.MAC
      A - Assemble MYFILE.MAC
```

Command Lines

```
   Menu Command               Expansion
 fsetfile 1 "Filename? "    SETFILE 1 "Filename? "
 eedit $fl                  EDIT MYFILE.MAC
 azex m80 $nl               ZEX M80 MYFILE
 ppage $fl                  PAGE MYFILE.MAC
```

Just as the HELP utility can take advantage of the highlighting facility provided in the Z3TCAP, so can VMENU. For those ZCPR3 Systems with a properly installed TCAP, VMENU will use the clear screen command to refresh the user's screen and highlighting can be enabled and disabled by embedding ^A (to turn on highlighting) and ^B (to turn off highlighting) into the *.VMN file.

It is recommended that when highlighting is turned on, it should be turned off in the same line for the sake of consistency and to improve appearance.

Example:

```
 #
          ^AThis is highlighted^B and this is not
 #
```

will appear with "This is highlighted" in a highlighted mode.

VMENU Variables

The $Pp variable is also available to the user under VMENU. $Pp returns information on the file currently being pointed to by the user on the screen. This variable has the following forms:

Form	*Expands Into*
$PF	FILENAME.TYP of the pointed-to file
$PN	FILENAME of the pointed-to file
$PT	TYP of the pointed-to file

For instance, if the file currently being pointed to is named MYFILE.TXT, then the command line:

```
ECHO FILENAME.TYP=$PF FILENAME=$PN TYP=$PT
```

will output:

```
FILENAME.TYP=MYFILE.TXT FILENAME=MYFILE TYP=TXT
```

Example:

```
-x
#
             Menu to Run M80 Assembler
                              Directory: $D$U
        E - Edit Pointed-to File
        P - Page Pointed-to File
        A - Assemble Pointed-to File
#
eedit $pf
azex m80 $pn
ppage $pf
##
```

Notes:

1. The E and P commands build command lines containing the full file name and type of the file being pointed to.
2. The A command shows the execution of ZEX. (V)MENU is a true ZCPR3 Shell, and, as such, ZEX commands can be issued from it and will run on top of it. The prompt for the ZEX command lines will be "VMenu>".

3. Assuming that MYFILE.MAC is being pointed to by the user, the following shows the expansion of the command lines for this example:

Menu Command	*Expansion*
eedit $pf	EDIT MYFILE.MAC
azex m80 $pn	ZEX M80 MYFILE
ppage $pf	PAGE MYFILE.MAC

The first entry in any (V)MENU file display is named "No File", and this entry, when pointed to by the user and expanded into the command line, is translated into a prompt for the user to input the name of a file. This feature is provided as a convenience to the user so that he will have the ability to easily specify new files which do not yet exist to the (V)MENU commands (such as for an editor command in which the user wants to create a new file).

If the pointer is at "No File" and the command line uses several references to the pointer (as in the ECHO command example above), then the user is prompted only once for the name of the file, and each reference derives its information from this name.

Closing Notes

As many commands as the printable ASCII character set (without lower-case letters and the (V)MENU command characters) will allow are permitted by (V)MENU. The text, however, for each menu must be able to fit on a screen with the file directory display at the top and the command prompt at the bottom. This means that the text cannot exceed 16 lines for VMENU or 20 lines for MENU.

(V)MENU fits in nicely to the ZCPR3 System of programs. The section on "Relationship of MENU and VMENU to the ZCPR3 System" (following) explains how (V)MENU and the other ZCPR3 programs work together.

The following ASCII characters may *not* be used as commands since they are used elsewhere:

```
<SPACE> # % , . < > * <DEL>
<Any Char Less than Space>
```

(V)MENU Programming Command Summary

Each (V)MENU command occupies only one line, and blank lines in the command group are not permitted. The command line is structured as follows:

```
l[o][command]
```

where:

l is the single character used to invoke the command; note that it may be upper- or lower-case.
o is an opening option, which is one of:
:nn -- go to Menu nn
! -- have (V)MENU wait when the command is finished
command is an optional ZCPR3 command; note that if the option is ":nn", then a command here makes no sense.

The (V)MENU commands are:

Command	*Function*
:nn	Goto Menu nn, where the first menu is Menu 1
!	Wait after command line is executed before processing the menu
"Prompt"	Prompt the user for input and accept it

The (V)MENU variables are:

Variable	*Expands to*
$D	Current Disk
$U	Current User
$Fn	FILENAME.TYP for System File n
$Nn	FILENAME for System File n
$Tn	TYP for System File n
$PF	FILENAME.TYP for Pointed-to File (VMENU only)
$PN	FILENAME for Pointed-to File (VMENU only)
$PT	TYP for Pointed-to File (VMENU only)
$$	$

Note: System Files can be defined by the SETFILE command.

The Highlighting Embedded Characters are:

```
^A Turn ON Highlighting
^B Turn OFF Highlighting
```

Note: It is recommended that if highlighting is turned on, it should be turned off in the same line.

The following ASCII characters may NOT be used as commands since they are used elsewhere:

```
<SPACE> # % , . < > * <DEL>
<Any Char Less than Space>
```

Relationship of MENU and VMENU to the ZCPR3 System

(V)MENU is installed by Z3INS. Like most of the ZCPR3 utilities, (V)MENU interacts with the system as a whole and cannot be used with systems other than ZCPR3. In particular, (V)MENU requires that the ZCPR3 Multiple Command Line Buffer and Shell Stack facilities be available to it and cannot run without them. (V)MENU invokes command lines via the Command Line Buffer and returns to itself thru the Shell Stack. VMENU (but not MENU) also uses the ZCPR3 System Files for some of its variables and the Z3TCAP facility for its screen manipulation (highlighting).

Also, CD (Change Directory) and STARTUP (or, ST for CD) can come into play with (V)MENU. When CD logs into a new directory, it looks for the file ST.COM and executes it if there is one. ST is simply STARTUP renamed, and STARTUP will load the Multiple Command Line Buffer with a command line and then terminate.

From the point of view of (V)MENU, the command loaded by ST could be (V)MENU. The effect of this is to automatically enter (V)MENU when the user employs CD to enter a given directory.

Hence, by using CD, a user can enter a directory and suddenly find himself in a menu instead of at the ZCPR3 command level. This is good for applications where a directory is set aside for a specific purpose and only certain operations are to be performed in it, such as cataloging disks or handling accounts.

Now that (V)MENU is running for the directory, a (V)MENU command could be another CD to another directory. Or it could simply be a DU: form. Example:

```
#
      A - Enter ZCPR Directory
      B - Enter A0:
#
acd zcpr:
ba0:
#
```

Here, if A is issued, then CD will move into ZCPR: and execute ST.COM if there is one there. If B is issued, the user is logged into A0:. (V)MENU is the next command in both cases (invoked as a Shell), so (V)MENU automatically reinvokes and looks for MENU.VMN. If it finds it, we are in another (V)MENU system, and, if it doesn't, we are back to ZCPR3 command level.

Under the A option, if CD finds ST.COM, ST will execute its function and, unless this function pops the Shell Stack (SHCTRL POP command), (V)MENU will reinvoke after it is complete.

Under the B option, we will run (V)MENU next and simply exit if a MENU.VMN file is not found.

VMENU (but not MENU) interacts heavily with the ZCPR3 System Files which are defined as a part of the ZCPR3 Environment Descriptor. There are four System Files, and three of them are used by VMENU for various purposes:

File	*Purpose*
2	Name of Current File
3	Name of Menu File
4	Name (containing wild cards) used to Select Files for VMENU File Display

System File 2 contains the name of the current VMENU file. By changing this name, a transient can cause the pointer of VMENU to point to some other file when VMENU is reinvoked.

System File 3 contains the name of the menu file which VMENU is using to derive menu displays and command from. By changing this entry, a transient can select different menu files dynamically.

System File 4 is used to indicate which files (such as *.TXT or *.*) are selected for display by VMENU when it is invoked. By changing this entry, the nature of the file display can be changed dynamically.

(V)MENU Error Messages

In order to make (V)MENU as small as possible, the error messages have been reduced to a minimum. (V)MENU provides a minimum indication that something is wrong and aborts.

The program (V)MENUCK is designed to tell the user more specifically what is wrong. (V)MENUCK is a *.VMN/*.MNU Syntax Checker, and it looks for all sorts of error conditions that can occur in a *.VMN/*.MNU file.

(V)MENU provides the following minimal error messages:

Message	*Meaning*
No Shell Stack	Shell stack not available
No Command Line	Command line buffer not available
Shell Stack Full	Shell stack is full
Shell Stack Size	Shell stack entries are too short for (V)MENU cmd line
File x.typ Not Found	Menu file not found
TPA Full	Memory is full
<Bell>	User command is in error
Structure Error	*.VMN File structure error
Password Error	Invalid password given (MENU only)

VMENUCK (version 1.0)
MENUCK (version 1.0)

Syntax:

VMENUCK dir:ufn <-- default file type is VMN
MENUCK dir:ufn <-- default file type is MNU

Function:

(V)MENUCK checks the syntax of a *.VMN/*.MNU file for the ZCPR3 Menu Shell, (V)MENU. (V)MENU is optimized for size and speed, and, in keeping it small, built-in diagnostics were reduced to the minimum. (V)MENUCK analyzes *.VMN/*.MNU files and provides informative diagnostics on any syntactical errors within them.

Options:

None.

Comments:

(V)MENUCK checks to see if the size of the *.VMN file is too large for the TPA available to the (V)MENU command. This is an additional check beyond the normal syntax check.

(V)MENUCK identifies the location of errors by line number. The first line in the file is line number 1.

Selected Error Messages:

Self-explanatory.

Examples of Use:

```
VMENUCK MYMENU     -- perform check on MYMENU.VMN
MENUCK MYMENU      -- perform check on MYMENU.MNU
```

6 Shell Subsystem

A number of different shells are supplied with ZCPR3, including VFILER and MENU. The principal shell processor is SH, which permits the user to use Named Variables. These are expanded in a manner similar to macros in his command lines. Two programs, SHDEFINE and SHVAR, allow the user to dynamically create Named Variables, and the SHFILE command allows Named Variable definitions to be grouped into sets of variables.

A conventional CP/M command line could look something like this:

```
ED MYFILE.TXT
```

Using SH, a ZCPR3 command line like

```
ED %WORKFILE
```

can be generated, and, as SH substitutes the definition of the variable WORKFILE when it interprets the command line, "ED %WORKFILE" could be expanded into "ED MYFILE.TXT" if WORKFILE=MYFILE.TXT. By changing the value of the variable WORKFILE, the meaning of the command "ED %WORKFILE" is correspondingly changed.

Once SH is invoked (by typing the command SH), any command typed by the user is passed thru SH first, expanded as required, and then, if the command is not an SH-resident command, the expanded command line is passed to ZCPR3 for processing.

SH variables may be nested to any depth. Recursion, however, should be avoided, and it is the responsibility of the user to ensure that recursion does not occur.

'%%' is interpreted by SH as a single '%'. For example, SH variables may be assigned as follows:

```
VAR1 = "ED %%VAR2" VAR2 = "MYFILE.TXT"
```

VAR1 is expanded as "ED %VAR2" which is, in turn, expanded to "ED MYFILE.TXT". This command is then passed to ZCPR3 for execution.

SH-Based Commands

There are three SH-resident commands:

SHCMT	switch SH to run in comment mode; in comment mode, all lines which do not begin with the character ! are treated as comments and flushed
SHECHO	with Echo enabled, all expanded command lines are printed to the user to show him what the line looked like after expansion
SHEXIT	SH is popped from the Shell Stack, enabling the next lower Shell for execution

Comment Mode. The normal prompt for SH is "DU:NAME>>", as opposed to "DU:NAME>" for ZCPR3. If the SHCMT (SH Comment) facility is enabled, the SH prompt becomes "DU:NAME;".

Any text issued by the user or from a command file (such as ZEX) will be processed as a comment unless the first character of the line is an exclamation mark (!), which is an indicator to process the command text that follows.

The command SHCMT switches to comment mode, and !SHCMT switches back.

SHECHO, SHEXIT, and ?. The SHECHO command is also a toggle, enabling and disabling the echo of command lines after all variables have been resolved.

The SHEXIT command causes the Shell Stack to be popped one level, which in turn causes the SH Shell to be terminated since it was on the top of the stack.

Both SHECHO and SHEXIT can be executed from SH Comment mode by prefixing these commands with an exclamation mark.

The ? command (a line beginning with a quesiton mark) invokes the built-in help facility of SH, which simply reminds the user of what the built-in commands are for SH.

Other Shells. SH is just the beginning of the possible ZCPR3 shell stack applications. MENU and VFILER are both shells also, and they execute like SH. All three shells can pass command lines to ZCPR3, have ZCPR3 execute these commands in its normal fashion (complete with the command-search hierarchy), and then return to the appropriate shells when done. A shell imposes a new initial command line interpretation on the input command line.

Selected Error Messages. "No Shell Stack" means that a Shell Stack has not been installed in the ZCPR3 System and SH cannot run.

"Shell Stack Full" means that there is not enough room on the Shell Stack for SH to push itself and SH cannot run.

"Shell Stack Entry Size" means that the shell stack entries are too short for SH to define the parameters it needs to control its operation. SH cannot run.

Potential Problems. Only one noted problem exists with SH. Certain ZCPR3-resident and SYSRCP-resident commands should be avoided. These commands include:

GO	because the TPA has been changed since the desired command executed
SAVE	same reason

SH tends to be a little slow in its loading. Unfortunately, SH has already been compressed as much as possible, so it will probably always take more time than simply invoking the ZCPR3 command processor directly.

Related Commands . The following commands are related to the SH shell and have an impact on its operation.

SHCTRL	This command can take direct control of the Shell Stack of ZCPR3. It can display the contents of the stack, pop the top-most shell off of it (thereby terminating the current shell), or clear the stack entirely (thereby terminating all shells on the stack).
SHDEFINE	This command can be used to interactively define a number of shell variables at one time. The user can display them and edit them as he desires.

SHFILE	This command defines the name of the shell variable file in the ZCPR3 Environment Descriptor. As many shell variable files as desired may be defined, each containing their own set of variables.
SHVAR	This command can be used to interactively define one shell variable at a time. It is slightly more convenient than SHDEFINE in some cases and can be used in command files and menus.

7 VFILER and File Maintenance

Overview of the VFILER File Maintenance Tool

VFILER (for Video FILER) version 3.0 gives the ZCPR3 user a specialized file manipulation utility that takes advantage of the special features of ZCPR3. It performs the same basic functions as DISK7, CLEANUP, WASH, and SWEEP, but has additional, ZCPR3-specific, commands and features that make VFILER more convenient than its predecessors for the ZCPR3 user. VFILER, unlike the tools mentioned above, is totally screen-oriented, being designed to run on a conventional CRT that supports cursor address, clear screen, and (optionally) erase to end of line.

VFILER significantly simplifies the user interface. An alphabetized listing of files is presented to the user along with a pointer. The user employs standard cursor movement commands to move the pointer up, down, right, or left until it is pointing to a file of interest. Once pointing to such a file, the user may then perform a number of operations on the file.

VFILER is invoked by a command line of the following form:

```
VFILER dir:filename.typ
```

where all parameters are optional. "DIR" is the directory to be made current; it may be any valid ZCPR3 directory reference (e.g., a mnemonic such as "ROOT:" or a DU form such as "B7:" or "12:"). "filename.typ" is an ambiguous file name which specifies an initial selection of the files to be processed by VFILER.

The reference for the ambiguous file name is stored in System File 4, and can therefore be dynamically changed by issuing a SETFILE command (e.g. SETFILE 4 afn) during the execution of VFILER. The result is to change the definition of this ambiguous file reference; the next time VFILER restarts execution, the new ambiguous file reference takes effect, and the files are selected accordingly.

The following description applies to VFILER 3.0. VFILER 3.0 will run only under ZCPR3, unless the Environment Descriptor is made internal to VFILER, in which case VFILER will be 1/4K larger but will run under earlier versions of ZCPR.

Installing VFILER

Installation of VFILER is quite simple, as is installation of most utilities under ZCPR3. Installation consists merely in providing VFILER with a pointer to the ZCPR3 Environment Descriptor (unless VFILER has been assembled to contain an Environment Descriptor, in which case the entire Environment Descriptor is necessary).

Z3INS can be used to install VFILER. To do this, create an INS file (call it VF.INS) containing the name of the VFILER.COM file on one line. Assuming that your system Environment Descriptor file is named SYS.ENV, issue the command:

```
Z3INS SYS.ENV VF.INS
```

This completes the normal installation of VFILER.

There are some customization equates at the front of the VFILER.MAC source file. The user will generally not need to change any of these, but may do so if he so desires. One such equate enables or disables the built-in documentation (help) feature. If this feature is disabled, any help reference will chain to a HLP file (VFILER.HLP) and VFILER.COM will be about 1K shorter. Enabling the built-in documentation feature provides the user with online help that is more concise but is available much more rapidly since it is memory-resident while VFILER is running.

VFILER Command Summary

```
--  Tagging Commands --     --------- File Operations -----------
  T - Tag File              C - Copy File D - Delete File
  U - Untag File            F - File Size R - Rename File
                            G - Group Copy/Delete/FSize/Tag/Untag

                    -- File Print & View --   --- User Functions ---
--  Cursor --       P - Print      V - View   0-9 - Execute # - Help
      ^E
      ^             -- Movement Commands --   --- Miscellaneous ---
  ^S <-+-> ^D        <SP> - File Forward      A - Toggle Alpha Sort
      v              <BS> - File Backward     H - Help File
      ^X              +   - Screen Forward    N - New DIR
                      -   - Screen Backward   S - Disk Status
--  Screen --         J   - Jump to a File    Z - ZCPR3 Command
   ^A Left            Q   - Refresh Screen   ^C - Exit
   ^F Right
```

```
    Movement Commands --

         ^E - Move Up (Wrap to Bottom)
         ^X - Move Down (Wrap to Top)
         ^D - Move Right (Wrap to First File of Next Line)
         ^S - Move Left (Wrap to Last File of Previous Line)
         ^F - Move Screen Right (Wrap to First Screen)
         ^A - Move Screen Left (Wrap to Last Screen)
```

The user's Z3TCAP entry may define four other single-character commands to conform to the arrow keys on his specific terminal. These commands will override the set described above if any conflicts exist (that is, if your down-arrow key generates a ^E, ^E will now mean Move Down in all cases).

Screen Left and Right make sense when there are too many files to fit on one screen. In this case, the files are broken into screen directories, and Screen Left and Right are used to move between them.

User Functions

The VFILER user may invoke up to ten extra commands that he has previously defined, by means of a set of user-definable functions. These functions are executed by striking a digit from 0 to 9. To implement user-defined functions, perform the following steps:

1. Create a file called VFILER.CMD containing your extended command set.
2. Place VFILER.CMD in some directory along your command search path.
3. When in VFILER, if you strike a digit or a pound sign (#) for help, VFILER searches along the path for the first VFILER.CMD file it finds and extracts the information from it.

Since VFILER searches for VFILER.CMD along the path, several VFILER.CMD files may be available for the user. For instance, if the path is $$ -> A$ -> A15, then VFILER will look for VFILER.CMD in the current directory, disk A/current user, and disk A/user 15. A general-purpose VFILER.CMD file may be placed in A15, and special-purpose VFILER.CMD files (e.g., for assembler language development, C development, word processing, etc.) may be placed in selected user areas on A. For instance, A7 might contain WordStar and be used for word processing, while B7 is the scratch area for text files. With this path, a user editing files in B7 will find WS in A7, VFILER.CMD (for word processing) in A7, and his system commands in A15.

If any of the extended commands require the selection of options, the user is prompted to supply them. When values for all of the mandatory options have been supplied, VFILER chains to the new command via the ZCPR3 Command Line Buffer feature, executes the command line generated, and returns. VFILER is a true shell under ZCPR3.

The structure of VFILER.CMD is quite simple. It can be created by any CP/M text editor, and consists of the following types of lines:

1. a command line, which begins with a digit (0-9) and contains the text of the command to be executed should that digit be typed by the user.
2. a help block, which is printed whenever the user types a pound sign (#); this block is denoted by a line which begins with a pound sign, and it extends to the end of the file.
3. a comment line, used for embedding explanatory comments, which are for reference purposes only and are not seen by the VFILER user.

Command Line. A command line consists of a digit, zero or more spaces (which are ignored), and the text of the command with embedded prompts for user input. These prompts are enclosed in single- or double-quotes (' or "). When VFILER executes these command lines, it prints the prompt contained within the quotes as they are encountered, and waits for the user to input a line of text (terminated by a RETURN). At this point the text is substituted in the command line containing the prompt. If a prompt extends to the end of a command line, it need not be terminated. For example the following line:

```
1 copy 'Source File? ' 'Destination Dir? '
```

defines user function 1. The user is prompted with "Source File?", he enters his text, it is substituted in the command line, he is prompted for "Destination Dir? ", he again enters a response, it is substituted, and the resulting command line is executed. In the above example, if the user responds with "myfile.txt" and "C0:", respectively, then the command line

```
copy myfile.txt C0:
```

is built. Note that spaces and other characters between the prompts are significant.

If the command line in VFILER.CMD contains the following:

```
1 mcopy 'Dest Dir? '='Source File? '
```

then the same responses will generate the command line

```
mcopy C0:=myfile.txt
```

Parameter Passing. Three parameters may be passed from VFILER into the command line being generated. These parameters and their symbols are:

Symbol	Parameter
%D	Current Disk Letter
%U	Current User Number (1 or 2 digits)
%F	Current File Name (pointed to by arrow)
%$	DU:FILENAME for Current File

If the user must insert a '%' character into the command line he is building, '%%' places one '%' into the line. For example:

```
echo Disk is %d, User is %u, File is %f
```

prints (if the user is in A15 and pointing to MYFILE.TXT):

```
Disk is A, User is 15, File is MYFILE.TXT
```

Help Block. The Help Block in the VFILER.CMD file is simply a block of text which extends from the pound sign (#) in the file to the end of the file. This help information is displayed to the user as one screen, and it is the responsibility of the person who writes the VFILER.CMD file to see that this body of text (including the line the pound sign is on) does not exceed 22 lines. Example:

```
# Help for Word Processing
    1 - Run WordStar
    2 - Run WordMaster
    3 - Run ROFF4
```

Comment Line. A comment line is any line which does not begin with a digit or a pound sign. The text of that line is the comment. It is not displayed to the VFILER user and is used only for reference to aid in maintaining the VFILER.CMD file. For example:

```
    ! This is a comment
      This is also a comment
```

Running ZEX from VFILER. Like all standard ZCPR3 Shells, VFIHER pupports the execution of the the ZEX command file processor on top of itself. If ZEX is running, VFILER will simply prompt ZEX for input rather than entering its normal screen-oriented display mode. In this way, a command executed from the VFILER.CMD file may invoke ZEX, and all of the ZEX command file processing will be performed before VFILER is reentered.

Sample VFILER.CMD File. To clarify the use of the user-defined functions, A sample VFILER.CMD file is shown below.

```
   ! VFILER Command File for Richard Conn
   1 xdir 'XDIR Options? '
   2 protect %D%U:%F 'PROTECT Attributes? '
   3 wm %$
   4 t2a
   5 echo Disk=%d  User=%u  File=%f  DU:FILENAME = %$
   #VFILER Command File for Richard Conn

   The following VFILER Macros are provided --

     1 - XDIR with Options
     2 - PROTECT Current File
     3 - Edit Current File
     4 - TERM III
     5 - Echo Current File and Text
```

Tagging Commands

T	Tag file for inclusion for mass file operations (group operations). The file remains tagged until either a disk log-in or 'U' is used to untag it. A 't' marker is placed by the tagged file name as a reminder the file is tagged for mass copy or mass delete.

U	Untag a file previously tagged for mass file operations. 'U' can be used to move cursor 'forward' for quick untagging of files. Logging-in drive again with 'N' also untags all files.

G T or U	Group (Mass) Tag or Untag. The user is prompted for the operation, and two of his options are T and U. If either operation is selected, tagging or untagging occurs automatically from the cursor position to the end of the screen on all files in this area. If the user then wishes to see the accumulated sizes of the tagged files, the G F (Group File Size) command may be issued.

File Display Commands

P	Print text file to CP/M list device (printer). Any keypress cancels.
V	View text file on console, with pagination and single-line turn-up. <CTRL-C> cancels function. <SPACE> advances to next line, and any other character advances the screen. Only ASCII characters are processed.
	Details on CRT and Printer sizes (number of lines on screen, number of lines on printer, etc.) are derived from the ZCPR3 Environment Descriptor and need not be of concern to the user. The command CPSEL can be used to select the CRT or Printer characteristics from the Environment Descriptor as desired.

File Operation Commands

C	Copy file to another DIR area with automatic CRC verification. The standard ZCPR3 DIR form is allowed, and a colon after the specification is optional. System reset occurs for disk change. The user is prompted to erase an already existing file on another drive or in other user areas. Before attempting a copy, check to see that there is enough room on the destination disk.
D	Delete file from disk. Reconfirmation is requested before the deletion is performed.
F	Display file size in kilobytes, rounded up to next disk allocation block.
G	Invoke Group (Mass) command. The three options of interest here are C (Copy all Tagged Files), D (Delete all Tagged Files), and F (File Size of all Tagged Files).
G C	Group copy of tagged files to another DU area. Auto-erase occurs if file(s) already exist(s). Prompts for desired DU area as with 'C' command. Group copy function can be repeated without re-tagging files.
G D	Group delete of tagged files. Prompts for approval: if the response is Y, deletion of all tagged files occurs without further user intervention; if the response is V, user is asked to approve each deletion before it is made. If the response is any other character, the operation aborts.
G F	Group file size summary. Adds up the file sizes of all tagged files and displays this sum.

R	Rename file on current drive. Only CP/M convention names permitted. Wild cards are not permitted. User is prompted for new file name.

Movement Commands

<SP>	Advance to next file name. Wraparound from last to first may occur. The WordStar ^D character or your right arrow key (if available in the Z3TCAP) perform the same function.
<BS>	Back up to last file name. Wraparound from first to last may occur. The WordStar ^S character or your left arrow key (if available in the Z3TCAP) perform the same function.
J	Jump to a file. Used to quickly jump to a specific file. User is prompted for a file name, and wild cards (? and *) may be used. User is positioned at first file which matches wild cards if found; user is positioned at first file in ring if not found.
Q	Refresh the screen. The current screen will be redisplayed.
+	Jump to Next Screen (if any). If there is more than one screen of files, the user is advanced to the next screen. If at the last screen, wraparound occurs to the first. ^F also performs this function if not overridden by arrow keys.
-	Jump to Last Screen (if any). Similar to + but in the opposite direction. Wraparound to last screen may occur. ^A also performs this function if not overridden by arrow keys.
Arrows	WordStar arrow key movement (if not overriden by arrow keys in Z3TCAP):

```
        ^E
         ^
   ^S <-+-> ^D
         v
        ^X
```

Miscellaneous VFILER Commands

A	Toggle Alpha Sort. This command reverses the sense of the sort of the current directory, reloads the directory, and refreshes the screen, having sorted it in the new sense. Sorting is done by file name and type or by file type and name.
H	Invokes external HELP Information. VFILER will chain to HELP and display the information in VFILER.HLP. VFILER checks to see if HELP can be found along path (external if available, internal if external path is not available) and does not attempt to chain if HELP.COM cannot be found.

N	Login new DU area for display and reset system for disk changes. Format of DU form is same as 'C' for copy.
S	Status of requested drive, shows remaining disk storage in kilobytes and number of files in current directory.
Z	Run any ZCPR3 Command Line. User will be prompted for command line, and VFILER will be reentered in same DU area as when command was executed. Command will execute in the original DU area as indicated by the prompt.
C	Exit to Operating System.
/ or ?	Print Command Summary (Short Help Info). VFILER may be assembled to omit built-in help, creating a VFILER which is about 1K shorter than a VFILER with the built-in help. If the built-in help is omitted, these commands chain to the VFILER.HLP file instead (see Installation, above).

8 DU3 Disk Utility

DU3 is intended for use on a ZCPR3 system, and is designed for installation with a minimum of trouble. In fact, in almost all cases, no changes to the source file should be necessary to get DU3 up and running. This is because DU3 uses the disk parameter block of CP/M to determine the characteristics of the disk environment.

DU3 is installed by running the ZCPR3 utility Z3INS on it. To perform its functions, DU3 needs only a pointer to the ZCPR3 Environment Descriptor. DU3 is assembled with VLIB, Z3LIB, and SYSLIB3.

The screen displays shown on the following pages are very close to the actual screen displays the user will see on his terminal when he runs DU3. The differences will be cosmetic in nature.

DU3 Command Summary. A command line may consist of one or more commands separated by commas; the commands are executed sequentially, in the order in which they appear. The only exceptions to this rule are the :ntext command (which stores the command line away as a macro) and the *nn command (which repeats the command line). The commands are listed below in the order in which experience shows that they are most frequently used.

Editing

E	Invoke Editor		

Positioning

Tn	Position to Track n (dec)	Sn	Position to Sec n
Gn	Position to Group n (hex)	G	Show position
+n	Advance to Next Sector	-n	Back up to Last Sec

Displaying

An-n	ASCII Dump	Hn-n	Hex Dump
Dn-n	ASCII and Hex Dump	Vn	View n Blocks
M	Display Disk Map	Mn	Display File in Group n

Data Modification

CAn text	Enter Text	CAn1-n2 char	Enter Char over Range
CHn vals	Enter Binary Values	CHn1-n2 val	Enter Value over Range

Disk Read/Write

R	Read Current Block	W	Write Current Block

Exiting DU3

X	Exit to ZCPR3	^C	Exit to ZCPR3

Macros

n	Exec Macro (0<=n<=9)	:nt	Define Macro n w/str t
:Pn	Print Macro n	:PA	Print All Macros
:P@	Print Prev Command		

Block/Group Queueing

<	Save Current Block in Temp	>	Get Saved Block
<B	Save Current Block on Queue	>B	Get Block from Queue
<G	Save Current Group on Queue	>G	Get Current Group
<Gn	Save Group N on Queue	>Gn	Get Group N

Queue Control

Q	Print Queue Statistics	QZ	Zero (Empty) Queue
QSf	Save Queue as File f		

Data Searching

Ff	Find File f	=str	Search for String
Un	Set User Area for Find		

Login/Disk Reset

Ld	Log in Disk	N	New Disk Reset

Printer Output

P	Toggle Printer

Command Manipulation

@	Exec Prev Command	*nn	Repeat Command Line

Statistics/Help

#	Display Disk Stats	?	Display Help Info

Halt/Sleep

!	Halt and Wait for User	Zn	Sleep n Seconds

DU3 Commands: Log In, New Disk, Stats

The following commands are discussed in this section:

L	Log in Disk
N	New Disk (Reset Disk System)
M	Map Disk Directory
U	Select User Number
#	Print Disk and Queue Statistics

Command: L[d]. The simple "L" command re-logs in the current disk. The user may pull out a disk, put in a new, and "L" just to log it in. The form "Ld" (where d is a valid drive letter) is used to log in a specific disk and permit the user to work on that

particular disk from then on.
Example (actual DU3 session):

```
DU3 B1? 1
DU3 B1? la
DU3 A1? lb
```

Command: N. This tells DU3 that the user just put in a new disk. For those BIOS implementations that need to be told specifically (Reset) that a disk change has been made, use this command every time a disk is changed while DU3 is running. Example:

```
DU3 B1?  n
```

Note: There was a significant delay before the prompt returned. A complete disk system reset took place.

Command: M[n]. Dumps a map of the group allocations for files. Mn shows which file is allocated to group "n". Example (actual DU3 session, edited):

```
DU3 B1? m
0010-0010  07 STD     .MSG 00 : 0011-0011  07 TALK    .SUM 00
0012-0012  07 Z2CON   .WSH 00 : 0013-0013  00 LDIR    .C   00
0014-0014  00 COMMAND .LBR 01 : 0015-0015  00 LDIR    .C   00
```

< Detail Left Out >

```
004B-004B  08 MASTER  .CAT 03 : 004C-0050  00 COMMAND .LBR 05
0051-0051  00 COMMAND .LBR 07 : 0052-0052  00 UNERA15 .COM 00
0053-0053  08 MENU    .CPR 00 : 0054-0057  00 COMMAND .LBR 07
Type Any Character to Continue or ^C to Abort -
DU3 B1? m54
0054-0057  00 COMMAND .LBR 07 :
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
```

The entries are divided as follows:

```
0010-0010  07 STD     .MSG 00 : 0011-0011  07 TALK    .SUM 00
  ^        ^  ^           ^
  |        |  Filename    Extent
  |        User Number
 Group Range
```

Command: Uu Logs user 'u' for next F (Find File) command.
Example (actual DU3 session):

```
DU3 B1? u7
DU3 B7? u1
```

Command: #. Prints the disk parameters:

- o Current Disk Drive
- o Size of Group in Blocks
- o Number of Groups on Disk
- o Number of System Tracks
- o Number of Tracks on Disk
- o Number of Sectors Per Track
- o Number of Directory Entries

Prints the queue statistics:

- o Size of Queue
- o Space Available

DU3 Commands: Movement

The following commands are discussed in this section:

G	Position to Group
S	Position to Sector
T	Position to Track
R	Read Block
W	Write Block
+	Advance to Next Logical Sector
-	Backup to Last Logical Sector

Command: G[nn]. Position to group nn and read block. If the form is simply "G", show the current position.

Example (actual DU3 session):

```
DU3 B1? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1

DU3 B1? g4
Group = 0004:00, Track = 122, Sector = 129, Physical Sector = 12

DU3 B1? g
Group = 0004:00, Track = 122, Sector = 129, Physical Sector = 12

DU3 B1? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
```

Command: Tnn and Snn. "Tnn" does a seek to track nn but does not read a block. "Snn" positions to sector nn on the current track and reads the block there. Example (actual DU3 session):

```
DU3 B1? t124
Group = 0015:00, Track = 124, Sector = 1, Physical Sector = 1

DU3 B1? s24
Group = 0015:17, Track = 124, Sector = 24, Physical Sector = 24
```

Command: R and W. R reads the block currently positioned to into memory. Note R (Read) is implicit in the G, +, and - commands, but NOT in the S and T commands.

W writes back the current block (NOTE: may not be used after an F command, as CP/M was used to find the file in the directory).

Examples:

```
DU3 B1? r
DU3 B1? w
```

Command: +[nn] and -[nn]. "+" advances 1 sector (if below track 2, this advances to next numerical sector and if 2 or more, advances based on the system's sector skewing algorithm, i.e. so + will get the next logical sector of the file). "-" backups up 1 sector in the same sense.

Note that "+" and "-" may take an amount: for example, +15 steps in 15 sectors. Note also that "-" issued at the first logical sector of the disk will wrap back to the last and "+" issued at last sector will wrap forward to the first.

Examples (actual DU3 session):

```
DU3 B1? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
DU3 B1? d
00 07535444 20202020 204D5347 00000002 |.STD     MSG....|
10 10000000 00000000 00000000 00000000 |................|
20 004C4449 52202020 20432020 00000038 |.LDIR     C ...8|
30 13001500 00000000 00000000 00000000 |................|
40 0843504D 55472020 20434154 01000046 |.CPMUG   CAT...F|
50 1C001E00 38003D00 41004300 45000000 |....8.=.A.C.E...|
60 00434F4D 4D414E44 204C4252 01000080 |.COMMAND LBR....|
70 14001600 17001800 19001A00 1B001D00 |................|

DU3 B1? +d
Group = 0000:01, Track = 122, Sector = 2, Physical Sector = 2
00 0754414C 4B202020 2053554D 00000049 |.TALK    SUM...I|
10 11002300 24000000 00000000 00000000 |..#.$...........|
20 00554E45 52413135 2041534D 00000060 |.UNERA15 ASM...'|
30 3E003F00 40000000 00000000 00000000 |>.?.@...........|
40 075A3249 4E532020 20575348 0000002A |.Z2INS   WSH...*|
50 21002500 00000000 00000000 00000000 |!.%.............|
60 075A3243 4F4E2020 20575348 0000003F |.Z2CON   WSH...?|
70 12002200 00000000 00000000 00000000 |..".............|

DU3 B1? +d
```

```
Group = 0000:02, Track = 122, Sector = 3, Physical Sector = 3
00 07445532 20202020 2042414B 00000057 |.DU3 BAK...W|
10 26002800 29000000 00000000 00000000 |&.(.)...........|
20 04535441 52545550 20C3CF4D 0000001E |.STARTUP COM....|
30 27000000 00000000 00000000 00000000 |'...............|
40 00434F4D 4D414E44 204C4252 03000080 |.COMMAND LBR....|
50 1F002000 30003100 32003300 34003500 |.. .0.1.2.3.4.5.|
60 00434F4D 4D414E44 204C4252 05000080 |.COMMAND LBR....|
70 36003700 39004C00 4D004E00 4F005000 |6.7.9.L.M.N.O.P.|

DU3 B1? +2
Group = 0000:04, Track = 122, Sector = 5, Physical Sector = 5

DU3 B1? d
00 084D4153 54455220 20434154 01000080 |.MASTER  CAT....|
10 2D002F00 3A003B00 3C004200 44004600 |-./.:.;.<.B.D.F.|
20 00464958 54455820 2041534D 0100001E |.FIXTEX  ASM....|
30 65006600 69006B00 6C000000 00000000 |e.f.i.k.l.......|
40 00554E45 52413135 20434F4D 00000007 |.UNERA15 COM....|
50 52000000 00000000 00000000 00000000 |R...............|
60 084D454E 55202020 20C35052 00000008 |.MENU    CPR....|
70 53000000 00000000 00000000 00000000 |S...............|

DU3 B1? -3d
Group = 0000:01, Track = 122, Sector = 2, Physical Sector = 2
00 0754414C 4B202020 2053554D 00000049 |.TALK    SUM...I|
10 11002300 24000000 00000000 00000000 |..#.$...........|
20 00554E45 52413135 2041534D 00000060 |.UNERA15 ASM...'|
30 3E003F00 40000000 00000000 00000000 |>.?.@...........|
40 075A3249 4E532020 20575348 0000002A |.Z2INS   WSH...||
50 21002500 00000000 00000000 00000000 |!.%.............|
60 075A3243 4F4E2020 20575348 0000003F |.Z2CON   WSH...?|
70 12002200 00000000 00000000 00000000 |..".............|
```

DU3 Commands: Searching

The commands for searching for data on the disk are:

Ffilename.typ – find all dir entries for file
=string – find next occurrence of string

Command: Ffilename.typ. Print directory for file "filename.typ". This command presents the directory entries for all extents of the indicated file. See the section on "Interpreting the DU3 Directory Display" for info on how to interpret the information presented. Example (actual DU3 session):

```
DU3 B1? fz80.mac
40 015A3830 20202020 204D4143 0000000E |.Z80     MAC....|
50 9A000000 00000000 00000000 00000000 |................|
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1

DU3 B1? ftest.txt
++ File Not Found ++
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
```

Command: =string. This command performs a search for the indicated ASCII text, starting at current sector. <xx> hex may be imbedded, or used alone: To find "IN 0FEH": =<db><fe>. Bit 7 is ignored unless <xx> is used. Note that, due to the parsing scheme of DU3, forms such as "+=string", which positions to the next sector and then starts the search, are allowed. Forms like "+2=string" are equally permitted. The search may be aborted by a ^C. Example:

```
DU3 B1? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
DU3 B1? =DU3
= at 24
Group = 0000:0E, Track = 122, Sector = 15, Physical Sector = 15
DU3 B1? d
00 07533130 30202020 20545854 0000000C |.S100    TXT....|
10 0C010000 00000000 00000000 00000000 |................|
20 07445532 20202020 2042414B 00000068 |.DU3     BAK...h|
30 2A006100 6D007800 00000000 00000000 ||.a.m.x.........|
40 015A3830 20202020 204D4143 0000000E |.Z80     MAC....|
50 9A000000 00000000 00000000 00000000 |................|
60 E5444953 4B4F5554 20434F4E 00000020 |eDISKOUT  CON...|
70 60000000 00000000 00000000 00000000 |'...............|

DU3 B1? +=DU3
Group = 0000:0F, Track = 122, Sector = 16, Physical Sector = 16
= at 64
Group = 0000:10, Track = 122, Sector = 17, Physical Sector = 17

DU3 B1? d
00 075A3243 4F4E2020 20575320 05000080 |.Z2CON   WS ....|
10 3D013E01 3F014001 41014201 43014401 |=.>.?.@.A.B.C.D.|
20 075A3243 4F4E2020 20575320 06000077 |.Z2CON    WS...w|
30 45014601 47014801 00000000 00000000 |E.F.G.H.........|
40 E5444953 4B4F5554 20434F4E 00000020 |eDISKOUT  CON...|
50 97000000 00000000 00000000 00000000 |................|
60 07445532 20202020 20484C50 01000006 |.DU3     HLP....|
70 98009900 9B009C00 9D000000 00000000 |................|

DU3 B1? +=DU3,d
Group = 0000:11, Track = 122, Sector = 18, Physical Sector = 18
= at 24
Group = 0000:11, Track = 122, Sector = 18, Physical Sector = 18
00 E547454E 494E5320 204D4143 01000080 |eGENINS  MAC....|
10 D700EF00 F000F400 F500F600 F700F800 |W.o.p.t.u.v.w.x.|
20 E5445532 20202020 2041534D 01000080 |eDU3     ASM....|
30 EC00ED00 EE00FA00 07010B01 0E011301 |l.m.n.z.........|
40 04445532 20202020 20C3CF4D 0000004E |.DU3     COM...N|
50 F100F200 F3000000 00000000 00000000 |q.r.s...........|
60 E547454E 494E5320 204D4143 02000039 |eGENINS  MAC...9|
70 F9000F01 00000000 00000000 00000000 |y...............|
```

DU3 Commands: Saving, Restoring, Queue

The following commands are discussed in this section:

<	Save Current Block
>	Restore Saved Block
<B	Save Current Block at Tail of Queue
>B	Load Current Block from Head of Queue
<G	Read and Save Group at Tail of Queue
>G	Copy Group from Head of Queue and Write
Q	Print Queue Statistics
QZ	Zero (Clear) Queue
QS	Save Queue as a File

Command: < and >. "<" saves current block in an internal save buffer. ">" copies the internal save buffer into the current block area (but does NOT write it out to disk).

Command: <B and >B. "<B" saves the current block onto the tail of the DU3 Queue. This Queue, a FIFO (First In-First Out) data structure, can be used to collect a number of blocks for later copy to a disk file or explicit placement somewhere on the disk. ">B" extracts the block at the head of the DU3 Queue and places it into the working buffer area.

Command: <G[nn] and >G[nn]. "<G" reads the current group and saves it on the tail of the DU3 Queue. The size of a group is dependent on the format of the disk, and DU3 automatically adjusts to the proper group size without the user having to worry about what it is. ">G" copies the group at the head of the DU3 Queue onto disk. If nn is specified (as in "<Gnn" or ">Gnn"), then the indicated group is read from or written to. If nn is not given, then the group the user is currently positioned to is affected.

Command: Q, QZ, and QSfile. Q reports the status of the DU3 Queue, namely how many blocks are stored in it and how much space remains. QZ zeroes (clears) the DU3 Queue. "QSfilename.typ" saves the DU3 Queue on disk in the current user area as the indicated file. Examples (actual DU3 session):

```
DU3 B1? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1

DU3 B1? d
00 07535444 20202020 204D5347 00000002 |.STD     MSG....|
10 10000000 00000000 00000000 00000000 |................|
20 004C4449 52202020 20432020 00000038 |.LDIR     C ...8|
30 13001500 00000000 00000000 00000000 |................|
40 0843504D 55472020 20434154 01000046 |.CPMUG   CAT...F|
50 1C001E00 38003D00 41004300 45000000 |....8.=.A.C.E...|
60 00434F4D 4D414E44 204C4252 01000080 |.COMMAND LBR....|
70 14001600 17001800 19001A00 1B001D00 |................|

DU3 B1? <

DU3 B1? ch0-7f e5

DU3 B1? d
00 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 |eeeeeeeeeeeeeeee|
10 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 |eeeeeeeeeeeeeeee|
20 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 |eeeeeeeeeeeeeeee|
30 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 |eeeeeeeeeeeeeeee|
40 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 |eeeeeeeeeeeeeeee|
50 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 |eeeeeeeeeeeeeeee|
60 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 |eeeeeeeeeeeeeeee|
70 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 |eeeeeeeeeeeeeeee|

DU3 B1? >,d
00 07535444 20202020 204D5347 00000002 |.STD     MSG....|
10 10000000 00000000 00000000 00000000 |................|
20 004C4449 52202020 20432020 00000038 |.LDIR     C ...8|
30 13001500 00000000 00000000 00000000 |................|
40 0843504D 55472020 20434154 01000046 |.CPMUG   CAT...F|
50 1C001E00 38003D00 41004300 45000000 |....8.=.A.C.E...|
60 00434F4D 4D414E44 204C4252 01000080 |.COMMAND LBR....|
70 14001600 17001800 19001A00 1B001D00 |................|
DU3 B1? q
** Queue Status Summary **
0 Blocks in Queue
249 Blocks Left in Queue
Address of Head of Queue: 3E00 Hex
Address of Tail of Queue: 3E00 Hex

DU3 B1? <g
Reading from Group 0000
32 Blocks in Queue
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1

DU3 B1? g1
Group = 0001:00, Track = 122, Sector = 33, Physical Sector = 33

DU3 B1? <g
Reading from Group 0001
64 Blocks in Queue
Group = 0001:00, Track = 122, Sector = 33, Physical Sector = 33

DU3 B1? q
** Queue Status Summary **
64 Blocks in Queue
185 Blocks Left in Queue
Address of Head of Queue: 3E00 Hex
Address of Tail of Queue: 5E00 Hex

DU3 B1? qsdir.sys
Queue Saved in File

DU3 B1? qz
** Queue Status Summary **
0 Blocks in Queue
249 Blocks Left in Queue
Address of Head of Queue: 3E00 Hex
Address of Tail of Queue: 3E00 Hex

DU3 B1? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
```

```
DU3 B1? <b
1 Blocks in Queue

DU3 B1? +<b
Group = 0000:01, Track = 122, Sector = 2, Physical Sector = 2
2 Blocks in Queue

DU3 B1? +<b
Group = 0000:02, Track = 122, Sector = 3, Physical Sector = 3
3 Blocks in Queue

DU3 B1? +<b
Group = 0000:03, Track = 122, Sector = 4, Physical Sector = 4
4 Blocks in Queue

DU3 B1? +2<b
Group = 0000:05, Track = 122, Sector = 6, Physical Sector = 6
5 Blocks in Queue
```

DU3 Commands: Display

The commands in this section are:

A	Display ASCII
D	Display ASCII and Hex
H	Display Hex
V	View as Text

Command: V[nn]. V views the current block as ASCII characters. The form "Vnn" views the indicated number of blocks starting at the current one.

Command: A, D, and H. Display a block or portion thereof, using the A command for ASCII characters only, D for both hexadecimal and ASCII, and H for hexadecimal numbers only.

```
D0-#7F   is the same as just D
D3-5
A20-#3F
```

See next section for examples.

DU3 Commands: Changing Data

The commands described in this section are:

CA	Change ASCII
CH	Change Hex

Examples are also given of the various display commands.

Command: CH and CA Allows the user to change the contents of the current block by specifying new values as hexadecimal numbers(CH) or as an ASCII character string (CA). Format is:

```
CHaddr val val val... --Change Hex data values in block
CAaddr char string... --Change ASCII data values in block
```

NOTE that an ASCII string may have hex values embedded in it:

```
ca0 OK<d><a><1a>
```

Use W to write changes to disk. Ranges may be specified; for example, <CHaddr-addr byte or CAaddr-addr byte> changes a range of bytes to the same value.

Examples (actual DU3 session):

```
DU3 B7? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1

DU3 B7? d
00 07535444 20202020 204D5347 00000002 |.STD     MSG....|
10 10000000 00000000 00000000 00000000 |................|
20 004C4449 52202020 20432020 00000038 |.LDIR     C ...8|
30 13001500 00000000 00000000 00000000 |................|
40 0843504D 55472020 20434154 01000046 |.CPMUG   CAT...F|
50 1C001E00 38003D00 41004300 45000000 |....8.=.A.C.E...|
60 00434F4D 4D414E44 204C4252 01000080 |.COMMAND LBR....|
70 14001600 17001800 19001A00 1B001D00 |................|

DU3 B7? d0-#f
00 07535444 20202020 204D5347 00000002 |.STD     MSG....|

DU3 B7? h0-#f
00 07535444 20202020 204D5347 00000002

DU3 B7? a0-#f
00 |.STD     MSG....|

DU3 B7? fdu2.hlp
20 07445532 20202020 20484C50 01000039 |.DU3     HLP...9|
30 28009800 99009B00 9C009D00 00000000 |(...............|
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
```

```
DU3 B7? g28
Group = 0028:00, Track = 125, Sector = 273, Physical Sector = 273

DU3 B7? d
00 496E766F 6B696E67 20445533 20616E64 |Invoking DU3 and|
10 20445533 20496E73 74616C6C 6174696F | DU3 Installatio|
20 6E0D0A44 55332043 6F6D6D61 6E642053 |n..DU3 Command S|
30 756D6D61 72790D0A 436F6D6D 616E6473 |ummary..Commands|
40 20666F72 204C6F67 67696E67 20446973 | for Logging Dis|
50 6B732061 6E642045 78616D69 6E696E67 |ks and Examining|
60 20446973 6B205061 72616D65 74657273 | Disk Parameters|
70 0D0A436F 6D6D616E 64732066 6F722050 |..Commands for P|

DU3 B7? v
Invoking DU3 and DU3 Installation
DU3 Command Summary
Commands for Logging Disks and Examining Disk Parameters
Commands for P
Group = 0028:00, Track = 125, Sector = 273, Physical Sector = 273

DU3 B7? v3
Invoking DU3 and DU3 Installation
DU3 Command Summary
Commands for Logging Disks and Examining Disk Parameters
Commands for Positioning and Reading Data
Commands for Searching for Data
Commands for Saving and Restoring Data
Commands for Viewing data
Commands for Altering Data
Commands for Manipulating Macros and the @ Command
Miscellanea
Examples of command use
Interpret
Group = 0028:02, Track = 125, Sector = 275, Physical Sector = 275

DU3 B7? g28
Group = 0028:00, Track = 125, Sector = 273, Physical Sector = 273

DU3 B7? d
00 496E766F 6B696E67 20445533 20616E64 |Invoking DU3 and|
10 20445533 20496E73 74616C6C 6174696F | DU3 Installatio|
20 6E0D0A44 55332043 6F6D6D61 6E642053 |n..DU3 Command S|
30 756D6D61 72790D0A 436F6D6D 616E6473 |ummary..Commands|
40 20666F72 204C6F67 67696E67 20446973 | for Logging Dis|
50 6B732061 6E642045 78616D69 6E696E67 |ks and Examining|
60 20446973 6B205061 72616D65 74657273 | Disk Parameters|
70 0D0A436F 6D6D616E 64732066 6F722050 |..Commands for P|

DU3 B7? ch0-10 0

DU3 B7? d
00 00000000 00000000 00000000 00000000 |................|
10 00445533 20496E73 74616C6C 6174696F |.DU3 Installatio|
20 6E0D0A44 55332043 6F6D6D61 6E642053 |n..DU3 Command S|
30 756D6D61 72790D0A 436F6D6D 616E6473 |ummary..Commands|
40 20666F72 204C6F67 67696E67 20446973 | for Logging Dis|
50 6B732061 6E642045 78616D69 6E696E67 |ks and Examining|
60 20446973 6B205061 72616D65 74657273 | Disk Parameters|
70 0D0A436F 6D6D616E 64732066 6F722050 |..Commands for P|

DU3 B7? call This is a test

DU3 B7? d0-#1f
00 00000000 00000000 00000000 00000000 |................|
10 00546869 73206973 20612074 6573746F |.This is a testo|
```

DU3 Commands: Macros

A Macro is a shorthand the user can employ to define a command sequence. Rather than having to type an involved command over and over again, the DU3 macro facility allows the user to assign this command sequence to a number (0 to 9) and then execute it by simply presenting this number as a command. The following commands are associated with this facility.

:ntext and n. ":n<text>" defines the text following the digit 'n' to be a Macro. As always, 0 <= n <= 9. The macro definitions may be created and redefined at will. If a macro has already been defined for the indicated number, it will be overwritten by the execution of this command. "n" (where 0 <= n <= 9) executes the indicated macro.

:Pn and :PA. ":Pn", where 0 <= n <= 9, prints the text of Macro Number n. ":PA" prints the text of all 10 macros.

@ and :P@. "@" executes the most recent command line that did not contain the "@" Command. This provides an easy way to repeat the last command line typed. For example:

```
    g0               <-- go to Group 0
```

```
ch0-7f e5,<    <-- Initialize the first block and Save
>,w,+          <-- Read in the Saved Block, Write it
                    out to disk, and advance to next
                    logical block
@              <-- Do the Previous Command Again
@              <-- And Again
```

":P@" prints the previous command line (without changing it). Examples (edited DU3 session):

```
DU3 B7? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1

DU3 B7? :1+,d0-#1f

DU3 B7? :p1
Macro Definitions --
1: +,d0-#1f

DU3 B7? 1
Group = 0000:01, Track = 122, Sector = 2, Physical Sector = 2
00 0754414C 4B202020 2053554D 00000049 |.TALK    SUM...I|
10 11002300 24000000 00000000 00000000 |..#.$...........|

DU3 B7? 1
Group = 0000:02, Track = 122, Sector = 3, Physical Sector = 3
00 07444953 4B4F5554 20434F4E 00000000 |.DISKOUT CON....|
10 00000000 00000000 00000000 00000000 |................|

DU3 B7? 1
Group = 0000:03, Track = 122, Sector = 4, Physical Sector = 4
00 04584449 52202020 20C3CF4D 00000054 |.XDIR    COM...T|
10 2C002E00 48000000 00000000 00000000 |,...H...........|

DU3 B7? g0,d0-#1f
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
00 07535444 20202020 204D5347 00000002 |.STD     MSG....|
10 10000000 00000000 00000000 00000000 |................|

DU3 B7? 1
Group = 0000:01, Track = 122, Sector = 2, Physical Sector = 2
```

```
00 0754414C 4B202020 2053554D 00000049 |.TALK SUM...I|
10 11002300 24000000 00000000 00000000 |..#.$...........|

DU3 B7? :pa
Macro Definitions --
0:

1: +,d0-#1f

2:
```

< Detail Left Out >

```
9:

DU3 B7? g0,d0-#1f,1,1,1
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
00 07535444 20202020 204D5347 00000002 |.STD    MSG....|
10 10000000 00000000 00000000 00000000 |................|
Group = 0000:01, Track = 122, Sector = 2, Physical Sector = 2
00 0754414C 4B202020 2053554D 00000049 |.TALK   SUM...I|
10 11002300 24000000 00000000 00000000 |..#.$...........|
Group = 0000:02, Track = 122, Sector = 3, Physical Sector = 3
00 07444953 4B4F5554 20434F4E 00000000 |.DISKOUT CON....|
10 00000000 00000000 00000000 00000000 |................|
Group = 0000:03, Track = 122, Sector = 4, Physical Sector = 4
00 04584449 52202020 20C3CF4D 00000054 |.XDIR   COM...T|
10 2C002E00 48000000 00000000 00000000 |,...H...........|

DU3 B7? @

Command --
g0,d0-#1f,+,d0-#1f,+,d0-#1f,+,d0-#1f
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1
00 07535444 20202020 204D5347 00000002 |.STD    MSG....|
10 10000000 00000000 00000000 00000000 |................|
Group = 0000:01, Track = 122, Sector = 2, Physical Sector = 2
00 0754414C 4B202020 2053554D 00000049 |.TALK   SUM...I|
10 11002300 24000000 00000000 00000000 |..#.$...........|

Group = 0000:02, Track = 122, Sector = 3, Physical Sector = 3
```

```
00 07444953 4B4F5554 20434F4E 00000000 |.DISKOUT CON....|
10 00000000 00000000 00000000 00000000 |................|
Group = 0000:03, Track = 122, Sector = 4, Physical Sector = 4
00 04584449 52202020 20C3CF4D 00000054 |.XDIR    COM...T|
10 2C002E00 48000000 00000000 00000000 |,...H...........|
```

DU3 Commands: Miscellaneous Command: ?. "?" gives a command summary and tells the user what the current values are for Processor Clock Speed and Lines per Page on CON: as well as the address for the Group Storage Buffer (where the DU3 Queue begins).

Command: *[nn]. "*nn" repeats the current command line (as entered so far) nn times. This command defaults to "forever" if nn is not specified. 'nn' may be 2 to 65535.

Command: !. "!" halts processing of commands, displays a continuation message to the user, and waits for the user to type any key. Typing a Control-C aborts command processing. This command is useful in stopping loops to give the user as much time as he wants to review the display.

Command: P. "P" toggles the printer switch on and off. It allows the user to turn on and off a recording of your console output.

Command: X. "X" exits back to ZCPR3.

Command: Z[nn]. "Znn" causes the program to sleep, or pause, and may be used to look at a dump quickly in a looping command line. Z is 1 sec. Znn is nn seconds on an MHz 8080. The processor speed is specified within the ZCPR3 Environment Descriptor.

Command: ^C. "^C" exits to ZCPR3 and causes a warm boot.

Examples (actual DU3 session):

```
DU3 B7? g0
Group = 0000:00, Track = 122, Sector = 1, Physical Sector = 1

DU3 B7? d0-#f,+,!,*
00 07535444 20202020 204D5347 00000002 |.STD    MSG....|
Group = 0000:01, Track = 122, Sector = 2, Physical Sector = 2

Type Any Character to Continue or ^C to Abort -
00 0754414C 4B202020 2053554D 00000049 |.TALK    SUM...I|
Group = 0000:02, Track = 122, Sector = 3, Physical Sector = 3

Type Any Character to Continue or ^C to Abort -
00 07444953 4B4F5554 20434F4E 00000000 |.DISKOUT CON....|
Group = 0000:03, Track = 122, Sector = 4, Physical Sector = 4

Type Any Character to Continue or ^C to Abort -
00 04584449 52202020 20C3CF4D 00000054 |.XDIR    COM...T|
Group = 0000:04, Track = 122, Sector = 5, Physical Sector = 5

Type Any Character to Continue or ^C to Abort -

DU3 B7? gl
Group = 0001:00, Track = 122, Sector = 33, Physical Sector = 33

DU3 B7? d0-#f,+,*3
00 E5482020 20202020 2042414B 00000004 |eH       BAK....|
Group = 0001:01, Track = 122, Sector = 34, Physical Sector = 34
00 E5535542 32202020 2042414B 0100007B |eSUB2    BAK...{|
Group = 0001:02, Track = 122, Sector = 35, Physical Sector = 35
00 E5434420 20202020 204D4143 00000047 |eCD      MAC...G|
Group = 0001:03, Track = 122, Sector = 36, Physical Sector = 36

DU3 B7? ^C
B7>
```

DU3 Command: Editor

DU3 contains a built-in, screen-oriented editor. This editor derives its screen-oriented functions from the ZCPR3 TCAP, so the DU3 Editor should be invoked only on ZCPR3 Systems that support a valid TCAP for the user's terminal.

The DU3 Editor is a complete subsystem under DU3 in its own right. It provides a variety of user-friendly editing features for the manipulation of data within the current sector (block) as well as allowing the user to issue any DU3 command line he desires, returning to the editor when it is completed.

The DU3 Editor presents a screen display to the user which is structured to include a line showing the contents of the sector at the cursor, several lines of hex/ASCII (similar to the D command output) which display the entire sector, a menu of commands, a cursor (which initially points to the first byte in a sector), and a command prompt.

Command: **E.** The command "E" invokes the editor. If any other commands follow E on the same line, these commands are flushed.

The user may employ the WordStar cursor movement commands to move the cursor about in the current sector. These commands are:

```
                  ^E = Cursor UP
                          ^
      ^S = Cursor LEFT <-+-> ^D = Cursor RIGHT
                          v
                  ^X = Cursor DOWN
```

^R refreshes the screen display.

The following commands are also available under the DU3 Editor:

A	Enter ASCII Text into block starting at cursor
H	Enter Hex/Dec Numbers into block starting at cursor
+	Advance to Next Logical Sector and Edit
-	Backup to Last Logical Sector and Edit
^W	Write the Current Sector to Disk
C	Issue Any DU3 Command Line
X	Exit to DU3
^C	Exit to ZCPR3

All of these commands are self-explanatory except for the A and H commands.

The A command enters ASCII text into the sector starting at the cursor position. In response to this command, the DU3 Editor will prompt the user for input. He may then type any text he wishes; upon striking the RETURN key, this text is entered literally into the sector. If it overflows the end of the sector, it is truncated. If the user wishes to embed hexadecimal values (such as 0D for Carriage Return), he may use the form "<hh>". Example:

```
this is a test<0D><0A>
```

The H command enters a group of hexadecimal and/or decimal values into the sector starting at the cursor position. Numbers, separated by spaces, are hexadecimal unless the the form "#nn" is used, in which case decimal values are entered. Example:

```
1 2 3 3A b7 #25
```

The Editor is one of the most powerful featues of DU3. It is highly recommended that the user experiment with it and become acquainted with its capabilities.

DU3 Examples

Multiple commands may be separated by ",".

Any valid command string may be placed as an operand of the original DU3 command, e.g.: A>DU3 G0,D,G2,=OK<D><A><1A>,D

Example: the following commands will fill the B disk directory with E5's:

```
lb            log in b drive
g0            position to dir.
ch0-7f e5     fill with e5
<             save the sector
>,w,+,/16     restore, write, next,
              repeat 16
```

This could be shortened to:

```
lb,g0,ch0-7f e5,<
>,w,+,/16
```

If we define the following two macros:

```
   Macro 0 --
     :0g0,ch0-7f e5,<
   Macro 1 --
     :1>,w,+,/16
```

To initialize the directory first on Drive A: and then on Drive B:, we could issue the following commands:

```
   La,0     <-- Log in A and Initialize first block
   1        <-- Perform write
   Lb,0     <-- Log in B and Initialize first block
   1        <-- Perform write
   n,0      <-- Declare New Disk and Initialize first block
   1        <-- Perform write
```

Directory Interpretation

The following diagram explains the format of a CP/M directory entry. Use DU3 with either the F (Find File) command or the D (Dump) command to display the directory sectors, which are located in groups 0 and 1 on a single density disk. Sample result of "FSID.COM" command:

```
First   40  00534944 20202020 20434F4D  0000003A  |.SID   COM...:
line    ||  |||                      |  ||    ||    |         |
        ||  ||^-----file name--------^  ||    ||   ^file name^
        ||  ||       in hex             ||    ||    in ASCII
        ||  ||                   extent-^^    ||
        ||  ||           file size in sectors-^^
        ||  ^^-00 = file active
        ||    E5 = file erased
        ^^-displacement of line in directory sector

Second 50  33343536 3738393A  00000000 00000000  |3456789:.......
line       |                                   |
           ^-----allocation group numbers-----^
```

le z9

9 Inside the ZCPR3 System Segments

A ZCPR3 System Segment is a package or data file which can be loaded by the LDR tool into memory for use by the ZCPR3 System. There are several types of ZCPR3 System Segments:

*.ENV Environment Descriptors

Section 2

Inside ZCPR3

"It was clear that a most powerful addition to any programming language would be the ability to define new higher level entities in terms of previously known ones, and then to call them by name. This would build the chunking right into the language. Instead of there being a determinate repertoire of instructions out of which all programs had to be explicitly assembled, the programmer could construct his own modules, each with its own name, each usable anywhere inside the program, just as if it had been a built-in feature of the language."
— D. R. Hofstadter, An Eternal Golden Braid

Inside ZCPR3

This section presents details of the internal operation of various components of the ZCPR3 System. In particular, the ZCPR3 command processor, ZEX, the system segments, shells, error handlers, screen-oriented utilities, and elements of the toolset in general are discussed.

This section is intended mainly for people who want to program utilities for ZCPR3. It is assumed that the reader is thoroughly familiar with CP/M and has experience in programming 8080/Z80 assembly language.

9 Inside the ZCPR3 Command Processor

This chapter describes the internal operations of the ZCPR3 Command Processor, the heart of the ZCPR3 System. The ZCPR3 Command Processor runs in place of the CP/M CCP, and it is located directly under the BDOS. To ensure compatibility with CP/M, the ZCPR3 Command Processor is 2K bytes in size or smaller and supports some structural and functional similarities. In order to pack all of the functions necessary to support a full ZCPR3 System into the Command Processor, it is mandatory that the Command Processor run on a Z80 microprocessor. The size reduction afforded by using jump-relative and other Z80-specific instructions in the Command Processor is significant, but, for the sake of those users who do not have Z80s, there is a flag at the front of the Command Processor which can select 8080 code. The resulting larger code forces a reduction in features, but very useful ZCPR3 systems can still be created for the 8080.

Operation of the ZCPR3 CP

Figure 1-1 illustrates the location and structure of the ZCPR3 Command Processor (CP). At various points in this book, the CP may also be referred to as a Command Processor Replacement (CPR), and the terms CP and CPR can be used interchangeably.

The main function of the ZCPR3 Command Processor (CP) can be described in a few high-level pseudo code instructions. The following is an English-like expression of the CP's operation:

```
initialize-environment
loop forever
     input (command_line)
     while command_line is not empty
          parse (next_command)
          resolve (next_command)
     end while
end loop
```

As the reader can see, the overall view of the ZCPR3 CP is quite simple. The major functions performed by the ZCPR3 CP are:

1. Initialize the environment
2. Input the command line
3. Parse the next command in the command line
4. Resolve the next command

We shall now discuss each of these functions in more detail.

Initialize_Environment

Environment initialization is discussed with reference to Listing 9-1, Code Sections 1 through 4.

The initialization of the the ZCPR3 CP environment can be expressed in the following pseudo-code:

```
entry
set_current_user_area
set_current_disk
set_DMA_address
set_running_SUBMIT_indicator
```

Entry. The ZCPR3 Command Processor supports two entry points (see Code Section 1). The first six bytes of the CP contain two jump instructions. The first jump is usually used for entry on cold boot, and the second jump is usually used on warm boot. In the original CP/M CCP, the first jump allowed a default command stored in the command line buffer internal to the CCP to be executed immediately and the second jump did not allow this command to be executed. This was useful in allowing CP/M to come up running a particular command line on cold boot (or even warm boot, for that matter, since the entry into the CCP was controlled by the cold boot and warm boot routines in the BIOS).

If ZCPR3 is implemented with an external Multiple Command Line buffer, these two entry points have the same meaning—to continue command line processing. Without an external Multiple Command Line buffer, the ZCPR3 CP contains a command line buffer immediately after these two jumps. This buffer is structured as follows:

```
ZCPR3_CP:                     ;beginning of ZCPR3 CP
          JMP  RUNCMD         ;first JMP instruction in CP
          JMP  NOCMD          ;second JMP instruction in CP

BUFLEN    EQU  80             ;recommended size of buffer
BUFSIZ:
          DB   BUFLEN         ;number of characters in buffer
CHRCNT:
          DB   0              ;input character count provided
                              ; thru the BDOS READLN routine
CMDLIN:
          DW   BUFLEN         ;buffer to contain command line
NXTCHR:
          DW   CMDLIN         ;pointer to character at which
                              ; to begin command processing
```

An external routine, such as the cold boot or warm boot routine, can store a command line (terminated by a binary 0) into this buffer at the symbol CMDLIN. Care should be taken not to exceed the character count at the symbol BUFSIZ (this count includes the terminating binary 0). Once the command line is stored, the routine may enter the ZCPR3 CP at the first JMP instruction and this command line will be executed. If the ZCPR3 CP is entered at the first JMP instruction, the pointer at NXTCHR will be set to the first byte in CMDLIN, and when the ZCPR3 CP is ready to accept input it will check to see if this line is empty (the pointer at NXTCHR is pointing to a binary zero), find such not to be the case, and parse and execute the command line. If the ZCPR3 CP is entered at the second JMP instruction, it will zero the byte at CMDLIN, thereby forcing an empty command line, set the pointer at NXTCHR to this zeroed byte at CMDLIN, and when the ZCPR3 CP is ready to accept input it will find an empty command line and obtain a new line from the running SUBMIT file ($$$.SUB) or the user (or ZEX if it is running).

The implementation of the ZCPR3 CP to include the Internal Command Line buffer is not recommended. Instead, it is recommended that an external Multiple Command Line buffer be used. This buffer, which must be initialized by the cold boot routine in the BIOS, has the following structure:

```
NXTCHR:
          DW    CMDLIN     ;pointer to first character of
                           ; command line
BUFLEN    EQU   200        ;recommended size of buffer
BUFSIZ:
          DB    BUFLEN     ;number of characters in buffer
CHRCNT:
          DB    0          ;input character count provided
                           ; thru the BDOS READLN routine
CMDLIN:
          DW    BUFLEN     ;buffer to contain command line
```

If this buffer is implemented, both entries to the ZCPR3 CP have exactly the same effect. The pointer at NXTCHR will be examined, and command line processing will resume at the next character in the buffer at CMDLIN. If the character pointed to by the address in NXTCHR is a binary 0, then the ZCPR3 CP will either input the next command line into the buffer at CMDLIN from a running SUBMIT file ($$$.SUB) if there is one, from the user or from ZEX, if ZEX is running. The pointer at NXTCHR will be reset to point to the first byte of CMDLIN and processing will resume.

Since the External Multiple Command Line Buffer is located in a "safe" place which is not affected by warm boots, this buffer provides the ability to support a command stream which does not change each time the system is warm booted. In the case of the Internal Command Line Buffer, each time a warm boot occurs the ZCPR3 CP would be reloaded from disk (a part of the definition of the warm boot process), and the internal command line buffer would be replaced by what was on disk. The warm boot routine in the BIOS may then choose to store a command in this buffer and enter the CP at the first JMP, but this is not nearly as efficient as simply having a command line buffer in a location in memory which is not affected by the warm boot process.

After the initial entry is performed through one of the two JMP instructions at the front of the ZCPR3 CP and the command line pointer is set (if an internal command line buffer is used), the initialization continues.

Set_current_user_area and Set_current_disk. The warm boot and cold boot routines must provide an input parameter to the ZCPR3 CP when they enter it at one of the two entry points. The C register contains the new current user area number in its upper four bits and the new current disk in its lower four bits. The routines set_current_user_area and set_current_disk derive their information from the C register and log the ZCPR3 CP into this directory. This becomes the user's current, or working, directory at cold or warm boot time. By the CP/M definition, memory location 4 (the *UD Byte*) in the buffer area in low memory contains the current UD (user area/disk) of the user, and the cold boot routine both sets this value initially and stores this value in the C register for entry into the ZCPR3 CP. The warm boot routine typically reads this value from location 4, stores it into the C register, and enters the ZCPR3 CP.

Set_DMA_address. This routine sets the address for the load of the next block from disk to 80H. This is done through BDOS calls.

Set_running_SUBMIT_indicator. This routine checks to see if a file named "$$$.SUB" exists on disk A and sets a flag to indicate the result. The BDOS disk reset function returns a code in the A register which is non-zero if the current user area on disk A contains a file beginning with a "$" character, and this clue is used to determine whether it is necessary to check disk A for a file named "$$$.SUB". If this clue is non-zero, the current user area on disk A (which is auto-logged by the first byte of the SUBFCB buffer) is checked for a $$$.SUB file, and the RNGSUB flag is finally set as a true indication that a $$$.SUB file exists on disk A, current user area. This flag is later used when the READBUF routine is used to input the next command line.

Input (Command_Line)

Once the initialization is performed, the ZCPR3 CP is ready to input a command line and process it. Under the recommended ZCPR3 CP configuration, however, the Multiple Command Line Buffer is available, so a command line may already be in existence and processing should continue with this line. Note that Code Section 4 of the initialization code branches to the entry point labelled RS1: after the initialization is complete; this is where the status of the command line buffer is determined. Listing 9-2, Code Sections 1-2, extracted from the ZCPR3 CP, shows the input command line processing routine.

The label RS1: is the entry point for processing the next command in the command line buffer. This point is reached when a cold or warm boot occurs and when a routine simply issues a RET instruction to return to the ZCPR3 CP after it has completed its function. RS1: is the principal entry point for command processing within the ZCPR3 CP.

Upon completion of any command under ZCPR3, the ZCPR3 CP performs two functions as it resumes control:

1. reset the DMA address
2. reset the current directory

These functions insure that the system is stable and any undesired states left by a process are negated. The process is invoked by a subroutine call ("CALL TPA" instruction, which may be translated into a call to some other location if the command is resolved within the ZCPR3 CP itself, within an RCP, or within an FCP) toward the end of the ZCPR3 CP source. If the process returns to the CP via a simple RET instruction, the command after this call is a command which restores the DMA address. If the process returns to the CP by means of a warm boot or cold boot, code in the front of the ZCPR3 CP also performs a restoration of the DMA address. Just after the label RS1: is a call to the DLOGIN routine which restores the current directory (defined at memory location 4).

Once this is complete, the pointer to the next character in the command line buffer is obtained (the LHLD NXTCHR instruction) and the command line from that point on is capitalized. This insures CP/M compatibility (all CP/M command lines are interpreted as upper case) and protects the system in the advent that the command line buffer was modified by the last command and the content was not capitalized in the process (see the code at CAPBUF: for details).

Now that the command line is secured, processing resumes at label RS2: by skipping over space characters and testing the last non-space character for (1) end of line and (2) abort. Additionally, if the Multiple Command Line Buffer is available, a test is made for the command separator character (a semicolon by recommendation), in which case the separator is skipped and RS2: is reentered at the character after the separator. The routine at RS2: is exited in one of two directions:

1. to the label RESTRT:, where a new command line is input
2. to the label RS3:, where the next command in the current command line is processed

A new command line is input into the Command Line Buffer (whether it is internal to the ZCPR3 CP or external as a Multiple Command Line Buffer) at the label RESTRT: (see Code Section 1).

A simple stack reset occurs at the label RESTRT:, and this is followed by the label RS0:, which is used as an entry point to flush comment lines (lines beginning with a semicolon).

The following steps are performed at the label RS0:

1. Indicate that the ZCPR3 CP is in normal execution mode. This indication is made to the input service routines by storing a zero value in byte 3 of the ZCPR3 Message Buffers. This message is then left on unless conditions elsewhere within the ZCPR3 CP change it, and its purpose is to pass information to the process which is next invoked which states that it was invoked as a conventional command. This message may be changed before the process is finally invoked, and this change will be explained as this commentary proceeds.

2. Indicate to ZEX (if it is running) that the ZCPR3 CP is now prompting for an input. This indication is made by storing the number 1 in byte 7 of the ZCPR3 Message Buffers. On the next call to input, if ZEX is running it will see this message and exit a hibernating state if it is in one.

3. The pointer to the next character in the command line is set to the first character position and this byte is zeroed. The storing of this zero is most important should the input routine be aborted and no input stored in the buffer. The zero will mark an end to the input line and a resumption of control by the ZCPR3 CP.

4. Input the command line via the READBUF routine. The READBUF routine accepts a command line from one of four sources in the order indicated: (1) the running SUBMIT file if one is in execution, (2) the shell stack if a command line is stored in it, (3) the ZEX monitor if it is running, and (4) the user. If the command line at the top of the shell stack is selected, the message at Z3MSG+3

(byte 3 of the ZCPR3 Message Buffers) is set to 1 to so indicate.

5. Once input is received, the ZEX message byte (byte 7 of the ZCPR3 Message Buffers) is cleared to zero. This indicates that the ZCPR3 CP prompt is no longer being presented.
6. Finally, the first character is checked to see if it is a semicolon, indicating a comment line, and RS0: is re-invoked if such is the case. If not, processing of the first command in the command line is begun by execution resuming at RS1:.

The READBUF routine is instrumental in selecting the source for the next command line and merits some further discussion. The source for this routine is shown in Listing 9-3, Code Sections 1-2.

Submit File Processing. The entry to READBUF begins with a test of the RNGSUB flag (which was set at initialization) to see if a $$$.SUB file is active. If this is the case, the following code extracts the next command line from it into the Command Line Buffer and returns to the caller of READBUF. The SUBMIT file execution prompt is displayed and the command line buffer is printed; a test for a user abort (via ^C) is also made and the abort is performed if such a command is given.

If the RNGSUB flag indicates that there is no $$$.SUB file or if there is no other line in the $$$.SUB file, execution resumes at the label RB1:. At RB1:, the SUBKIL routine is called and any running SUBMIT file (as indicated by RNGSUB) is deleted.

Shell Processing. A test is then made to see if the shell stack (if present) contains any elements. A binary zero in the first shell stack element is used to indicate an empty shell stack.

Shells are fundamental to the design of the ZCPR3 System, and their execution is performed like any other command line by the ZCPR3 CP. The only difference is that the command line stored on the shell stack is executed whenever the ZCPR3 CP decides to accept a new input line (and $$$.SUB is not running), and the ZCPR3 CP leaves a different value in its Command Status message to indicate that it has executed a shell. See Code Section 2 for details.

If the shell stack is determined not to be empty, then the command line stored in the top element is copied into the Command Line Buffer. The SHSIZE constant is used to determine how many characters long the shell command line is for this copy. Once the command line has been copied, the ZCPR3 CP Command Status message at byte 3 (Z3MSG+3) is set to 1 to indicate that a shell has been invoked. This message is used to communicate directly with the shell to let it know that it had been invoked as a shell as opposed to a user-specified command. See the technical detail section on shells later in the book for more detail.

If the command source is determined to not be either a running SUBMIT file or a shell, then the routine at RB2: is entered. The conventional ZCPR3 CP prompt is printed (CPRMPT is output as opposed to SPRMPT by the SUBMIT file processor) and input is accepted directly into the Command Line Buffer through the BDOS input line routine.

ZEX Input. If the ZEX monitor is running (the ZEX command has been issued at some time previously) and the ZEX command file stored in memory is not empty, then ZEX will start supplying input characters through its BIOS intercept routine. If the ZEX monitor is not running, normal input will be provided from the user.

Parse (Next_Command)

The command line is input and residing in the Command Line Buffer at this time, and the time has come to parse the non-empty command line after the label RS3:. The code for this parse is invoked by the following simple call:

```
;
; PARSE COMMAND LINE PTED TO BY HL
;
     CALL     PARSER          ;PARSE ENTIRE COMMAND LINE
```

Command Line. The ZCPR3 Command Line can contain one of several forms of commands at any one time:

1. The command line can be empty
2. The command line can be a comment line: ; comment text
3. The command line can contain one command: command
4. The command line can contain a sequence of commands, separated by semicolons: command_1;command_2;...

The ZCPR3 command consists of the following general structure:
verb dir:filename.typ dir:filename.typ text

Examples of ZCPR3 commands:

```
WS MYFILE.TXT
XDIR ROOT:*.TXT
RENAME TEXT:FILE1.TXT=F1.TXT
```

The ZCPR3 CP Parser parses each command into buffers as per the following extended form of the CP/M CCP parsing standard:

Verb : The verb is stored in the File Name field of the External FCB if one exists. The File Type field is set to COM. Any process can determine its name by examining the File Name field of the External FCB.

Dir:filename.typ: The first token after the verb is assumed to be in this general format, and it is taken apart as per the CP/M convention and stored in the FCB at memory location 5CH. If the prefix is DIR: or DU:, the disk reference is placed in the first byte (disk A = 1) and the user area reference is placed in the 13th byte (FCB+13), which is the S1 field. If the disk is not referenced, then the first byte is set to 0 (at FCB) to indicate the current disk. If the user area is not referenced, the 13th byte (at S1) is set to the current user area. The field at FCB+16 (at D0) is set to zero (a side effect of the general algorithm and not of general utility).

Second Dir:filename.typ: The second token after the verb is also assumed to be in this general format, and it is parsed in the same manner as described for the first token. Again, the disk reference and user area reference are stored in relative positions 0 (at FCB2) and 13 (at FCB2+13). The field at FCB2+16 (at CR) is zeroed, and this is significant since it assures that a subsequent open of the file positions the FCB to the first block of the file.

Tail: The part of the command starting with the space delimiter immediately following the verb is stored into the buffer area starting at 81H. A character count is stored at 80H, and the byte immediately following the last byte of the tail is set to a binary 0 value. This is as per the CP/M convention. This buffer is set during the CALLPROG routine as opposed to the PARSER routine; PARSER simply saves the location of the first byte of the tail.

Note 1: If either of the two tokens following the verb are omitted in the command, the file name and file type fields are filled with spaces as per the CP/M convention. The disk and user areas are zero-filled in this case.

Note 2: In resolving the Dir: prefix in either of the two tokens, the PARSER routine can be made to scan for DU forms before DIR forms or vice-versa. This provides a means of resolving conflicts between the two forms, such as when a disk named C is available and a directory named C is also available. If the DU form is resolved first, then a directory named C can never be referenced. If the DIR form is resolved first, any references to the disk C must include the user area number to distinguish from the directory named C. A password check is also done when a DIR: reference is resolved.

Resolve (Next_Command)

The final step in the main ZCPR3 CP functional loop is the resolution of the parsed command. The code for this step is shown in Listing 9-4.

During the execution of the READBUF routine, the message at Z3MSG+3 (the ZCPR3 Command Status message) was set to a 1 if the command line associated with a shell was selected. The same message was cleared to a zero before the READBUF routine was called.

Shell Command. The first step in resolving a command is to see if the source of the command is the shell stack, and, if so, to proceed to command resolution for CP-resident, RCP-resident, and transient commands at the label RS4:. This enables the shell to be invoked regardless of the Flow Command State; if the Flow Command State is FALSE, the ZCPR3 Shell must still be invoked so that a command line can be presented. If this is not allowed for, if the Flow Command State ever becomes FALSE during the execution of a shell, the system will deadlock. Therefore, this test for a shell invocation is performed immediately.

FCP Resolution. If the command is not a shell, the command scanner is run on the FCP command table. It is important that FCP-based commands are always executed because they control the Flow Command State. If a command (FCP-based or otherwise) ever sets the Flow Command State to FALSE, the only way to set it back to TRUE is to execute an FCP-based command. This is why the test for an FCP-based command is performed before the Flow Command State is checked.

Command Scanner. The Command Scanner in the ZCPR3 CP is used to resolve FCP-based, RCP-based, and ZCPR3 CP-based commands. Each of these are implemented as *Command Packages*, where a package is a collection of commands in one logical grouping. Like a package in the Ada programming language, a Command Package in ZCPR3 is divided into two parts—the *visible section* and the *hidden section.* The visible section consists of a table of command names and the addresses of the routines which implement these commands. The hidden section contains the bodies of the commands, where each command acts as a COM file would, extracting the information it needs, such as its verb, the two FCBs, and the command tail, from the buffers set up by the PARSER routine. Unlike the CP/M CCP, all parsing is done in one routine, the PARSER, and every command in the ZCPR3 System acts just as a

COM file would.

RCPs and FCPs. RCPs and FCPs are both implementations of Command Packages and they support a similar structure. This structure is:

```
COMMAND_PACKAGE:
                          DB  'Z3xCP'     ; Z3RCP or Z3FCP
COMMAND_PACKAGE+5:
                          DB  NAME_SIZE   ; number of bytes
                                          ; in each command
                                          ; name
                          DB  'CMD1'      ; command name
                                          ; (NAME_SIZE=4 here)
                          DW  CMD1_ADR    ; command address
                          DB  'CMD2'
                          DW  CMD2_ADR
                              ...
                          DB  0           ; 0 terminates table
                              ...
CMD1_ADR:
                          ...             ; body of command
                          RET             ; or JMP 0 to end
CMD2_ADR:
                          ...             ; body of command
                          RET             ; or JMP 0 to end
```

Flow Command State. Returning to the main train of thought, the FCP-resident command table has now been scanned. If a match is found, the ZCPR3 CP executes the command immediately. If not, a test is done to determine if the current Flow State is TRUE. The bytes at Z3MSG+1 (the current IF level) and Z3MSG+2 (the Active IF indicator) are checked. If the current IF level is 0 (the byte at Z3MSG+1 is zero), then command execution is allowed and we proceed. If not, a check has to be done to see that the current IF level is TRUE. This is done by ANDing the bytes at Z3MSG+1 and Z3MSG+2. The current IF level ANDed with the Active IF indicator tells the ZCPR3 CP whether the current Flow Command State is TRUE or not. If not TRUE, the command is flushed and processing is resumed at RS1. If TRUE, the command processing resumes at RS4.

At RS4, a check is made to see if the command was prefixed with a directory reference; that is, the form of the verb was:

```
DIR:VERB or DU:VERB
```

If so, the resident command check is bypassed and the search for a COM file begins immediately. This feature allows the ZCPR3 CP-resident and RCP-resident Command Packages to be bypassed. If not, a check is first done to see if the command is in the

ZCPR3 CP-resident Command Package and, if not, if the command is in the RCP-resident Command Package. Finally, if both of these tests fail, a search is done for the command as a COM file.

COM File Processing

No description of the ZCPR3 Command Processor (CP) would be complete without a description of the algorithm used to locate and load COM files. This code is implemented near the end of the CP in Section 5I. Listing 9-5, Code Sections 1-3 shows this code.

The entry point at COMDIR (see Code Section 1 below) is entered if a prefix is attached to the verb in the command:

```
DIR:VERB or DU:VERB
```

Upon entry at COMDIR, a check is made to see if the command name (verb) is blank. If so, then a command to log into a new directory was given:

```
DIR: or DU:
```

The directory reference is processed immediately. Note that a check is made of the selected user area (at FCBDN+13) to ensure that the user area is within range. With directories under ZCPR3, thirty-two user areas (numbered 0-31) can be referenced, but only sixteen (numbered 0-15) can be logged into. If the user area reference is valid, the directory is logged into and command processing is resumed at RS1.

The resolution of a COM file begins at the label COM: (see Code Section 2). The TPA is selected as the load address, and the MLOAD routine is called to locate the COM file and load it into memory at the load address (which is in HL upon entry to MLOAD). If MLOAD returns, then the COM file has been loaded successfully and the ZCPR3 CP is logged into the current directory. A new line is issued, the command line tail is stored in the buffer at 80H, the DMA address is set to 80H, and the TPA is called.

Invoking Any Command

Once a command has been resolved, regardless if it is ZCPR3 CP-resident, RCP-resident, FCP-resident, or transient, the routine CALLPROG (see Code Section 2) is used to invoke it. RCP-resident and transient commands are invoked at the label CALLPROG, which issues a new line before beginning, and CP-resident and FCP-resident commands are invoked at the label CALLP, which does not issue a new line. The CALLPROG/CALLP routine is called with the execution address in HL, and the function of CALLPROG is as follows:

1. Set the execution address for a subroutine call.
2. Store the command tail into the buffer at 80H. The PARSER routine marked the beginning of the command tail in the TAILSV buffer during the parse.
3. Set the DMA address to 80H.
4. Call the routine. If the routine is a transient, a call is made to the TPA (100H). Otherwise, a call is made to the address returned by CMDSCAN, the ZCPR3 CP Command Table Scanner.

5. Restore the DMA address and resume command line processing at the label RS1. This point is reached if the command routine completed with a simple RET instead of a JMP 0 (Warm Boot) or other technique.

During the execution of MLOAD, if the COM file is not found, then either the CMDRUN facility will be invoked or the ERROR routine will be invoked (if CMDRUN is not available). In the case of CMDRUN, the entire command is passed as a command tail to the CMDRUN utility.

Path Analysis

The MLOAD routine in the ZCPR3 CP loads a COM file, and, in so doing, it follows the Command Search Path in searching for the COM file. If the MINPATH option is not selected, the MLOAD routine simply follows the symbolic path, resolving each symbol as it goes. This can lead to wasted effort if the same directory is encountered more than once as the path is resolved. For instance, if the path is "A$ A15" and the user is logged into A15, then the path resolves into "A15 A15", and the directory A15 is logged into twice.

The MINPATH option eliminates this potential inefficiency by passing over the symbolic path once, building an absolute path into the buffer at MPATH. For each symbolic element encountered, MLOAD/MINPATH checks to see if the absolute path already contains that directory reference and does not include it if such is the case. When complete, it is this absolute path which is scanned during the search for the COM file.

Error Handling

If the command cannot be satisfactorily resolved, the ERROR routine is executed to determine how to process the current error condition. Listing 9-6, Code Sections 1-2, shows the code for the ERROR processor.

The ERROR routine first terminates any running SUBMIT file since, if there is one, the $$$.SUB was the source of the error. It then issues a new line and checks to see if the error was caused by an attempt to invoke a shell. If Z3MSG+3 (the ZCPR3 Command Status message) is set to 1, then the last command was invoked as a shell by the ZCPR3 CP. If the error was caused by trying to invoke a shell, the shell stack is cleared (at ERRSH) and command processing is restarted (at RESTRT).

If a shell was not involved, the Error Handler indicator message at Z3MSG is checked. If this message is non-zero, then an Error Handler was installed. If zero, then there is no Error Handler and the command in error is simply echoed followed by a question mark (the CP/M convention) and the command processing is restarted from scratch (at RESTRT).

If an Error Handler is available, then the ZCPR3 Command Status message (at Z3MSG+3) is set to 2 to indicate that an Error Handler is being invoked, the pointer to the command in error is set at Z3MSG+4 and Z3MSG+5 (the error line address), and the command line at Z3MSG+10H (the Error Handler command line) is invoked as any other command would be at RS1. This, naturally, assumes that the Error Handler itself can be resolved. It is best to place all Error Handlers in the ROOT directory (at the end of the Command Search path) to ensure that the ZCPR3 CP does not enter an infinite loop in trying to invoke a non-existent Error Handler.

ZCPR3 Command Processor— Wrapup

This concludes the technical presentation on the ZCPR3 Command Processor (CP). As the reader has seen, the ZCPR3 CP performs a high degree of interaction with the messages of the ZCPR3 System, and all must be in tune for the system to work correctly. Of most importance is that the ZCPR3 messages must be properly initialized at Cold Boot and that all of the tools must contain a proper pointer to the ZCPR3 Environment Descriptor in order to function correctly. From the ZCPR3 Environment Descriptor, the details of the ZCPR3 System, including the ZCPR3 Message Buffers, can be determined, and the tools can adapt themselves to any ZCPR3 System configuration in this manner.

The following chapters in this section address other key items in the ZCPR3 System. Emphasis is placed on how various classes of tools, such as Error Handlers and Shells, interact with the ZCPR3 System.

```
    ORG      CPRLOC
;
;  ENTRY POINTS INTO ZCPR3
;
```

< Comments omitted >

```
ENTRY:
    JMP      CPR             ;Process potential default command
    JMP      CPR1            ;Do NOT process potential default command

;
;**** Section 1 ****
; BUFFERS ET AL
;
;  **** 1. INPUT COMMAND LINE AND DEFAULT COMMAND
;
    IF       MULTCMD             ;MULTIPLE COMMANDS ALLOWED?
```

< Comments omitted >

```
NXTCHR       EQU    Z3CL            ;NXTCHR STORED EXTERNALLY (2 byte
BUFSIZ       EQU    NXTCHR+2        ;BUFSIZ STORED EXTERNALLY (1 byte
CHRCNT       EQU    BUFSIZ+1        ;CHRCNT STORED EXTERNALLY (1 byte
CMDLIN       EQU    CHRCNT+1        ;CMDLIN STORED EXTERNALLY (long)
BUFLEN       EQU    Z3CLS           ;LENGTH OF BUFFER
;
   ELSE
```

< Comments omitted >

```
BUFLEN      EQU    80               ;MAXIMUM BUFFER LENGTH
BUFSIZ:
   DB       BUFLEN        ;MAXIMUM BUFFER LENGTH
CHRCNT:
   DB       0             ;NUMBER OF VALID CHARS IN COMMAND LINE
CMDLIN:
   DB       '             '      ;DEFAULT (COLD BOOT) COMMAND
   DB       0                    ;COMMAND STRING TERMINATOR
   DS       BUFLEN-($-CMDLIN)+1 ;TOTAL IS 'BUFLEN' BYTES
;
NXTCHR:
   DW       CMDLIN        ;POINTER TO COMMAND INPUT BUFFER
;
   ENDIF                  ;MULTCMD
;
```

Listing 9-1. ZCPR3 CP: Initialize Environment, Code Section 1

```
;
;  **** 2. FILE TYPE FOR COMMAND
;
COMMSG:
    COMTYP                ;USE MACRO FROM Z3HDR.LIB
;
    IF      SUBON         ;IF SUBMIT FACILITY ENABLED ...
;
;  **** 3. SUBMIT FILE CONTROL BLOCK
;
SUBFCB:
    DB      1             ;DISK NAME SET TO DEFAULT TO DRIVE A:
    DB      '$$$'         ;FILE NAME
    DB      '     '
    SUBTYP                ;USE MACRO FROM Z3HDR.LIB
    DB      0             ;EXTENT NUMBER
    DB      0             ;S1
SUBFS2:
    DS      1             ;S2
SUBFRC:
    DS      1             ;RECORD COUNT
    DS      16            ;DISK GROUP MAP
```

```
SUBFCR:
    DS      1               ;CURRENT RECORD NUMBER
;
    ENDIF           ;SUBON
;
;  **** 4. COMMAND FILE CONTROL BLOCK
;
    IF      EXTFCB NE 0   ;MAY BE PLACED EXTERNAL TO ZCPR3
;
FCBDN       EQU     EXTFCB        ;DISK NAME
FCBFN       EQU     FCBDN+1       ;FILE NAME
FCBFT       EQU     FCBFN+8       ;FILE TYPE
FCBDM       EQU     FCBFT+7       ;DISK GROUP MAP
FCBCR       EQU     FCBDM+16      ;CURRENT RECORD NUMBER
;
    ELSE                    ;OR INTERNAL TO ZCPR3
;
FCBDN:
    DS      1               ;DISK NAME
FCBFN:
    DS      8               ;FILE NAME
FCBFT:
    DS      3               ;FILE TYPE
    DS      1               ;EXTENT NUMBER
    DS      2               ;S1 AND S2
    DS      1               ;RECORD COUNT
FCBDM:
    DS      16              ;DISK GROUP MAP
FCBCR:
    DS      1               ;CURRENT RECORD NUMBER
```

Listing 9-1. ZCPR3 CP: Initialize Environment, Code Section 2

```
;
    ENDIF           ;EXTFCB
;

;
;  **** 5. LINE COUNT BUFFER
;
```

```
    IF      LTON
PAGCNT:
    DB      NLINES-2      ;LINES LEFT ON PAGE
    ENDIF          ;LTON
;
;  **** 6. RESIDENT COMMAND TABLE
;   EACH TABLE ENTRY IS STRUCTURED AS FOLLOWS:
;   DB      'NAME';NCHARS LONG
;   DW      ADDRESS ;ADDRESS OF COMMAND
;
CMDTBL:
    DB      NCHARS;SIZE OF TEXT IN COMMAND TABLE
    CTABLE         ;DEFINE COMMAND TABLE VIA MACRO IN Z3HDR FILE
    DB      0      ;END OF TABLE
;


;
;**** Section 2 ****
; ZCPR3 STARTING POINTS
;
; START ZCPR3 AND DON'T PROCESS DEFAULT COMMAND STORED IF
; MULTIPLE COMMANDS ARE NOT ALLOWED
;
CPR1:
;
    IF      NOT MULTCMD    ;IF MULTIPLE COMMANDS NOT ALLOWED
;
    XRA     A              ;SET END OF COMMAND LINE SO NO
                                  ; DEFAULT COMMAND
    STA     CMDLIN         ;FIRST CHAR OF BUFFER
;
    ENDIF          ;NOT MULTCMD
```

Listing 9-1. ZCPR3 CP: Initialize Environment, Code Section 3

< Comments Omitted >

```
CPR:
    LXI     SP,STACK       ;RESET STACK
```

```
;
        IF      NOT MULTCMD     ;ONLY ONE COMMAND PERMITTED
        LXI     H,CMDLIN        ;SET PTR TO BEGINNING OF COMMAND LINE
        SHLD    NXTCHR
        ENDIF           ;NOT MULTCMD
;
        PUSH    B
        MOV     A,C             ;C=USER/DISK NUMBER (SEE LOC 4)
        RAR                     ;EXTRACT USER NUMBER
        RAR
        RAR
        RAR
        ANI     0FH
        STA     CURUSR          ;SET USER
        CALL    SETUSR          ;SET USER NUMBER
        CALL    RESET           ;RESET DISK SYSTEM
;
        IF      SUBON           ;IF SUBMIT FACILITY ENABLED
;
        STA     RNGSUB          ;SAVE SUBMIT CLUE FROM DRIVE A:
;
        ENDIF           ;SUBON
;
        POP     B
        MOV     A,C             ;C=USER/DISK NUMBER (SEE LOC 4)
        ANI     0FH             ;EXTRACT CURRENT DISK DRIVE
        STA     CURDR           ;SET IT
        CNZ     LOGIN           ;LOG IN DEFAULT DISK IF NOT ALREADY LOGGED I
        CALL    SETUD           ;SET USER/DISK FLAG
        CALL    DEFDMA          ;SET DEFAULT DMA ADDRESS
;
        IF      SUBON           ;CHECK FOR $$$.SUB IF SUBMIT FACILITY IS ON
;
        LXI     D,SUBFCB        ;CHECK FOR $$$.SUB ON CURRENT DISK
RNGSUB  EQU     $+1             ;POINTER FOR IN-THE-CODE MODIFICATION
        MVI     A,0             ;2ND BYTE (IMMEDIATE ARG) IS THE RNGSUB FLAG
        ORA     A               ;SET FLAGS ON CLUE
        CNZ     SEAR1
        STA     RNGSUB          ;SET FLAG (0=NO $$$.SUB)
;
```

```
	ENDIF		;SUBON
;
	JR	RS1		;CHECK COMMAND LINE FOR CONTENT
```

Listing 9-1. ZCPR3 CP: Initialize Environment, Code Section 4

```
;
; PROMPT USER AND INPUT COMMAND LINE FROM HIM
;
RESTRT:
	LXI	SP,STACK	;RESET STACK
;
; READ INPUT LINE FROM USER OR $$$.SUB
;
RS0:
;
	IF	Z3MSG NE 0
	XRA	A		;SET NO OUTPUT MESSAGE
	STA	Z3MSG+3		;ZCPR3 COMMAND STATUS
	INR	A		;SET ZCPR3 INPUT PROMPT
	STA	Z3MSG+7		;ZEX MESSAGE BYTE
	ENDIF		;Z3MSG NE 0
;
	LXI	H,CMDLIN	;SET POINTER TO FIRST CHAR IN COMMAND LINE
	SHLD	NXTCHR		;POINTER TO NEXT CHARACTER TO PROCESS
	MVI	M,0		;ZERO OUT COMMAND LINE IN CASE OF WARM BOOT
	PUSH	H		;SAVE PTR
	CALL	READBUF		;INPUT COMMAND LINE FROM USER (OR $$$.SUB)
;
	IF	Z3MSG NE 0
	XRA	A		;NORMAL PROCESSING RESUMED
	STA	Z3MSG+7		;ZEX MESSAGE BYTE
	ENDIF
;
	POP	H		;GET PTR
	MOV	A,M		;CHECK FOR COMMENT LINE
	CPI	COMMENT		;BEGINS WITH COMMENT CHAR?
	JRZ	RS0		;INPUT ANOTHER LINE IF SO
```

Listing 9-2. ZCPR3 CP: Command Line Input, Code Section 1

```
;
; PROCESS INPUT LINE; NXTCHR PTS TO FIRST LETTER OF COMMAND
;
RS1:
    LXI     SP,STACK        ;RESET STACK
;
; RETURN TO CURRENT DIRECTORY AND POINT TO NEXT CHAR IN COMMAND LI
;
    CALL    DLOGIN          ;RETURN TO CURRENT DIRECTORY
    LHLD    NXTCHR          ;PT TO FIRST CHAR OF NEXT COMMAND
    PUSH    H               ;SAVE PTR
;
; CAPITALIZE COMMAND LINE
;
CAPBUF:
    MOV     A,M             ;CAPITALIZE COMMAND CHAR
    CALL    UCASE
    MOV     M,A
    INX     H               ;PT TO NEXT CHAR
    ORA     A               ;EOL?
    JRNZ    CAPBUF
    POP     H               ;GET PTR TO FIRST CHAR IN LINE
;
; SET POINTER FOR MULTIPLE COMMAND LINE PROCESSING TO FIRST CHAR
; OF NEW CMND
;
RS2:
    CALL    SKSP            ;SKIP OVER SPACES
    ORA     A               ;END OF LINE?
    JRZ     RESTRT
    CPI     CTRLC           ;ABORT CHAR?
    JRZ     RESTRT
;
    IF      MULTCMD         ;MULTIPLE COMMANDS ALLOWED?
    MOV     A,M             ;GET FIRST CHAR OF COMMAND
    CPI     CMDSEP          ;IS IT A COMMAND SEPARATOR?
    JRNZ    RS3
    INX     H               ;SKIP IT IF IT IS
    JR       RS2
    ENDIF          ;MULTCMD
;
RS3:
    SHLD    NXTCHR          ;SET PTR TO FIRST CHAR OF NEW COMMAND LINE
    SHLD    CURCMD          ;SAVE PTR TO COMMAND LINE FOR ERROR RETURN
```

Listing 9-2. ZCPR3 CP: Command Line Input, Code Section 2

```
;
; INPUT NEXT COMMAND TO CPR
;   This routine determines if a SUBMIT file is being processed
; and extracts the command line from it if so or from the user's console
;
READBUF:
;
    IF      SUBON           ;IF SUBMIT FACILITY IS ENABLED, CHECK FOR IT
;
    LDA     RNGSUB          ;SUBMIT FILE CURRENTLY IN EXECUTION?
    ORA     A               ;0=NO
    JRZ     RB1             ;GET LINE FROM CONSOLE IF NOT
    LXI     D,SUBFCB        ;OPEN $$$.SUB
    PUSH    D               ;SAVE DE
    CALL    OPEN
    POP     D               ;RESTORE DE
    JRZ     RB1             ;ERASE $$$.SUB IF END OF FILE AND GET CMND
    LDA     SUBFRC          ;GET VALUE OF LAST RECORD IN FILE
    DCR     A               ;PT TO NEXT TO LAST RECORD
    STA     SUBFCR          ;SAVE NEW VALUE OF LAST RECORD IN $$$.SUB
    CALL    READ            ;DE=SUBFCB
    JRNZ    RB1             ;ABORT $$$.SUB IF ERROR IN READING LAST REC
    LXI     D,CHRCNT        ;COPY LAST RECORD (NEXT SUBMIT CMND) TO CHRCN
    LXI     H,TBUFF         ;  FROM TBUFF
    MVI     B,BUFLEN        ;NUMBER OF BYTES
    CALL    LDIR
    LXI     H,SUBFS2        ;PT TO S2 OF $$$.SUB FCB
    MVI     M,0             ;SET S2 TO ZERO
    INX     H               ;PT TO RECORD COUNT
    DCR     M               ;DECREMENT RECORD COUNT OF $$$.SUB
    LXI     D,SUBFCB        ;CLOSE $$$.SUB
    CALL    CLOSE
    JRZ     RB1             ;ABORT $$$.SUB IF ERROR
    CALL    PROMPT          ;PRINT PROMPT
    MVI     A,SPRMPT        ;PRINT SUBMIT PROMPT TRAILER
    CALL    CONOUT
    LXI     H,CMDLIN        ;PRINT COMMAND LINE FROM $$$.SUB
    CALL    PRIN1
    CALL    BREAK           ;CHECK FOR ABORT (ANY CHAR)
    RNZ                     ;IF NO ^C, RETURN TO CALLER AND RUN
    CALL    SUBKIL          ;KILL $$$.SUB IF ABORT
    JMP     RESTRT          ;RESTART CPR
```

Listing 9-3. ZCPR3 CP: READBUF, Code Section 1

```
;
; INPUT COMMAND LINE FROM USER CONSOLE
;
RB1:
    CALL    SUBKIL          ;ERASE $$$.SUB IF PRESENT
;
    ENDIF        ;SUBON
;
;  IF SHELL STACKS ARE IMPLEMENTED, CHECK FOR CONTENT AT THIS TIM
;
    IF      SHSTK NE 0
;
    LXI     H,SHSTK         ;PT TO STACK
    MOV     A,M             ;CHECK FIRST BYTE
    CPI     ' '+1           ;SEE IF ANY ENTRY
    JRC     RB2             ;GET USER INPUT IF NONE
;
    ENDIF         ;SHSTK NE 0
;
    IF      (SHSTK NE 0) OR (Z3MSG NE 0)
;
RUNBUF:
    LXI     D,CMDLIN        ;PT TO FIRST CHAR OF COMMAND LINE
    MVI     B,SHSIZE        ;COPY SHELL LINE INTO COMMAND LINE BUF
    CALL    LDIR            ;DO COPY
    XCHG                    ;HL PTS TO END OF LINE
    MVI     A,1             ;SAY SHELL WAS INVOKED
    STA     Z3MSG+3         ;Z3 OUTPUT MESSAGE
    JR      RB3             ;STORE ENDING ZERO AND EXIT
RB2:
;
    ENDIF          ;(SHSTK NE 0) OR (Z3MSG NE 0)
;
    CALL    PROMPT          ;PRINT PROMPT
    MVI     A,CPRMPT        ;PRINT PROMPT TRAILER
    CALL    CONOUT
    MVI     C,0AH           ;READ COMMAND LINE FROM USER
    LXI     D,BUFSIZ        ;PT TO BUFFER SIZE BYTE OF COMMAND LINE
    CALL    BDOS
;
; STORE ZERO AT END OF COMMAND LINE
;
    LXI     H,CHRCNT        ;PT TO CHAR COUNT
    MOV     A,M             ;GET CHAR COUNT
    INX     H               ;PT TO FIRST CHAR OF COMMAND LINE
    CALL    ADDAH           ;PT TO AFTER LAST CHAR OF COMMAND LINE
RB3:
    MVI     M,0             ;STORE ENDING ZERO
    RET
```

Listing 9-3. ZCPR3 CP: READBUF, Code Section 2

```
    IF      Z3MSG NE 0
    LDA     Z3MSG+3         ;GET COMMAND STATUS
    CPI     1               ;SHELL?
    JZ      RS4
    ENDIF          ;Z3MSG NE 0
;
; IF IFON AND FCP AVAILABLE, TRY TO RUN FROM FCP
;
    IF      IFON AND (FCP NE 0)
    LXI     H,FCP+5         ;PT TO COMMAND TABLE
    CALL    CMDSCAN         ;SCAN TABLE
    JZ      CALLP           ;RUN IF FOUND (NO LEADING CRLF)
    ENDIF          ;IFON AND (FCP NE 0)
```

```
;
; IF IFON, THEN CHECK FOR RUNNING IF AND FLUSH COMMAND LINE IF ENABLED
;
    IF      IFON
    LXI     H,Z3MSG+1       ;PT TO IF BYTE
    MOV     A,M             ;GET IT
    ORA     A               ;SEE IF ANY IF
    JRZ     RS4             ;CONTINUE IF NOT
    INX     H               ;PT TO IF ACTIVE BYTE
    ANA     M               ;SEE IF CURRENT IF IS ACTIVE
    JRZ     RS1             ;SKIP IF NOT
    ENDIF           ;IFON
RS4:
;
; IF DIR: PREFIX, HANDLE AS COM FILE
;
COLON       EQU     $+1     ;FLAG FOR IN-THE-CODE MODIFICATION
    MVI     A,0             ;COMMAND OF THE FORM 'DIR:COMMAND'?
    ORA     A               ;0=NO
    JNZ     COMDIR          ;PROCESS AS COM FILE IF DIR: FORM
;
; CHECK FOR RESIDENT COMMAND
;
    CALL    CMDSER          ;SCAN FOR CPR-RESIDENT COMMAND
    JZ      CALLP           ;RUN CPR-RESIDENT COMMAND WITH NO LEADING C
;
; CHECK FOR RESIDENT COMMAND PACKAGE
;
    IF      RCP NE 0
    LXI     H,RCP+5         ;PT TO RCP COMMAND TABLE
    CALL    CMDSCAN         ;CHECK FOR RCP
    JZ      CALLPROG
    ENDIF
;
; PROCESS AS COM FILE
;
    JMP     COM             ;PROCESS COM FILE
```

Listing 9-4. ZCPR3 CP: Command Resolution

```
;
;Section 5I
;Command: COM file processing
;Function:  To load the specified COM file from disk and execute it
;Forms:  <command line>
```

< Comments Omitted >

```
COMDIR:
   IF      DRVPREFIX
;
   LDA     FCBFN            ;ANY COMMAND?
   CPI     ' '              ;' ' MEANS COMMAND WAS 'DIR:' TO SWITCH
   JRNZ    COM              ;NOT <SP>, SO MUST BE TRANSIENT OR ERROR
;
;  ENTRY POINT TO SELECT USER/DISK VIA DIR: PREFIX
;
   IF      WDU   ;WHEEL FACILITY?
   CALL    WHLCHK           ;CHECK FOR WHEEL BYTE
   ENDIF         ;WDU
;
   LDA     FCBDN+13         ;GET SELECTED USER
   CPI     16               ;OUT OF RANGE?
   JNC     ERROR
   LXI     D,FCBDN          ;PT TO FCB
   CALL    FCBLOG           ;LOG INTO DU
   LDA     TEMPUSR          ;GET TEMPORARY USER
   STA     CURUSR           ;SET CURRENT USER (MAKE PERMANENT)
   LDA     TEMPDR           ;GET SELECTED DISK
   ORA     A                ;IF 0 (DEFAULT), NO CHANGE
   JRZ     COMDR
   DCR     A                ;ADJUST FOR LOGIN
   STA     CURDR            ;SET CURRENT DRIVE
COMDR:
   CALL    SETUD            ;SET UD BYTE
   JMP     RS1              ;RESUME COMMAND LINE PROCESSING
;
   ENDIF         ;DRVPREFIX
```

Listing 9-5. ZCPR3 CP: COM File Processing, Code Section 1

```
;
;   PROCESS COMMAND
;
COM:
;
    IF      CMDRUN          ;COMMAND RUN FACILITY AVAILABLE?
    MVI     A,0FFH          ;USE IT IF AVAILABLE (MLOAD INPUT)
    ENDIF         ;CMDRUN
;


;
; SET EXECUTION AND LOAD ADDRESS
;
    LXI     H,TPA           ;TRANSIENT PROGRAM AREA
    PUSH    H               ;SAVE TPA ADDRESS FOR EXECUTION
    CALL    MLOAD           ;LOAD MEMORY WITH FILE SPECIFIED IN CMD LI
    POP     H               ;GET EXECUTION ADDRESS; FALL THRU TO CALLP
;
; CALLPROG IS THE ENTRY POINT FOR THE EXECUTION OF THE LOADED
;   PROGRAM; ON ENTRY TO THIS ROUTINE, HL MUST CONTAIN THE EXECUTION
;   ADDRESS OF THE PROGRAM (SUBROUTINE) TO EXECUTE
;
CALLPROG:
    CALL    CRLF            ;LEADING NEW LINE
CALLP:
    SHLD    EXECADR         ;PERFORM IN-LINE CODE MODIFICATION
;
; COPY COMMAND TAIL INTO TBUFF
;
TAILSV      EQU     $+1     ;POINTER FOR IN-THE-CODE MODIFICATION
    LXI     H,0             ;ADDRESS OF FIRST CHAR OF COMMAND TAIL
    LXI     D,TBUFF         ;PT TO TBUFF
    PUSH    D               ;SAVE PTR
    MVI     B,0             ;SET COUNTER
    INX     D               ;PT TO FIRST CHAR
TAIL:
    MOV     A,M             ;GET CHAR
    CALL    TSTEOL          ;CHECK FOR EOL
    JRZ     TAIL1
    STAX    D               ;PUT CHAR
    INX     H               ;PT TO NEXT
    INX     D
    INR     B               ;INCREMENT COUNT
    JR      TAIL
TAIL1:
    XRA     A               ;STORE ENDING ZERO
    STAX    D
    POP     H               ;GET PTR
    MOV     M,B             ;SAVE COUNT
```

Listing 9-5. ZCPR3 CP: COM File Processing, Code Section 2

```
;
; RUN LOADED TRANSIENT PROGRAM
;
    CALL    DEFDMA          ;SET DMA TO 0080
;
; EXECUTION (CALL) OF PROGRAM (SUBROUTINE) OCCURS HERE
;
EXECADR     EQU     $+1     ;CHANGE ADDRESS FOR IN-LINE CODE MODIFICATIO
    CALL    TPA             ;CALL TRANSIENT
;
; RETURN FROM EXECUTION
;
    CALL    DEFDMA          ;SET DMA TO 0080, IN CASE PROG CHANGED IT
    JMP     RS1             ;RESTART CPR AND CONTINUE COMMAND PROCESSING
```

Listing 9-5. ZCPR3 CP: COM File Processing, Code Section 3

```
;
; ERROR PROCESSOR
;
ERROR:
;
    IF      SUBON           ;IF SUBMIT FACILITY IS ON
;
    CALL    SUBKIL          ;TERMINATE ACTIVE $$$.SUB IF ANY
;
```

```
    ENDIF           ;SUBON
;
    CALL    CRLF            ;NEW LINE
;
    IF      Z3MSG NE 0      ;MESSAGES ENABLED?
;
    LDA     Z3MSG+3         ;WAS ERROR CAUSED BY NO SHELL?
    ANI     1               ;BIT 0 SAYS ZCPR3 TRIED TO RUN A SHELL
    JRNZ    ERRSH           ;ABORT SHELL
    LDA     Z3MSG           ;GET ERROR HANDLER MESSAGE
    MOV     B,A             ;... IN B
    ORA     A               ;FLUSH AND RESUME?
    JRZ     ERR0
    MVI     A,2             ;SET ERROR FLAG
    STA     Z3MSG+3         ;IN SHELL STATUS BUFFER
    LHLD    CURCMD          ;PT TO BEGINNING OF ERROR
    SHLD    Z3MSG+4         ;SAVE IN MESSAGE
    LXI     H,Z3MSG+10H     ;PT TO COMMAND LINE
    SHLD    NXTCHR          ;NEXT CHARACTER TO EXECUTE
    JMP     RS1             ;RUN CONTENTS OF BUFFER
;
; CLEAR SHELL STACK AND RESTART COMMAND PROCESSING
;
ERRSH:
;
    IF      SHSTK NE 0      ;IF SHELL STACK AVAILABLE
    XRA     A               ;CLEAR SHELL STACK
    STA     SHSTK
    ENDIF
;
    JMP     RESTRT          ;RESTART PROCESSING
```

Listing 9-6. ZCPR3 CP: Error Processor, Code Section 1

```
ERR0:
;
   ENDIF            ;Z3MSG NE 0
;
CURCMD      EQU     $+1     ;POINTER FOR IN-THE-CODE MODIFICATION
    LXI     H,0             ;PT TO BEGINNING OF COMMAND LINE
```

```
ERR1:
    MOV     A,M             ;GET CHAR
    ORA     A               ;END OF LINE?
    JRZ     ERR2
    CALL    CONOUT          ;PRINT COMMAND CHAR
    INX     H               ;PT TO NEXT CHAR
    JR      ERR1            ;CONTINUE
ERR2:
    CALL    PRINT           ;PRINT '?'
    DB      '?'+80H
ERR3:
    JMP     RESTRT          ;RESTART CPR
```

Listing 9-6. ZCPR3 CP: Error Processor, Code Section 2

10 Inside the ZCPR3 System Segments

A ZCPR3 System Segment is a package or data file which can be loaded by the LDR tool into memory for use by the ZCPR3 System. There are several types of ZCPR3 System Segments:

*.ENV	Environment Descriptors
*.FCP	Flow Command Packages
*.RCP	Resident Command Packages
*.IOP	Input/Output Packages
*.NDR	Named Directory Data Files
*.Z3T	Terminal (TCAP) Data Files

Environment Descriptor

The main purpose of the Environment Descriptor is to provide information about the ZCPR3 System environment to the ZCPR3 tools in a convenient, transportable form so that the tools can be moved from one ZCPR3 System to another at the binary level. The only installation requirement for transportability is that the toolset be installed via the Z3INS tool; this process takes less than five minutes for the installation of over 70 tools. The ZCPR3 Environment Descriptor, which includes a Z3TCAP entry for the user's terminal, provides a significant extension over the conventional CP/M facilities for porting programs at the binary level from one computer system to another.

Flow and Resident Command Packages

Like the package feature of the Ada programming language, a Command Package is divided into two parts: the visible section, which defines the names and addresses of the commands in the package, and the hidden section, where the commands are implemented. Both Flow Command Packages and Resident Command Packages are written in the same general format:

```
     DB    'Z3xCP'    ; Z3RCP for Resident Command Package
                      ; Z3FCP for Flow Command Package
     DB    NAME_SIZE  ; number of characters in command name
     DB    'CMD1'     ; name of command (NAME_SIZE chars long)
     DW    CMD1_ADR   ; address of command
     ...
     DB 0             ; end of table
CMD1_ADR:
     ...              ; code which implements command
     RET              ; simple RET instruction is adequate
                      ; when command is completed
```

The routine CMDSCAN in the ZCPR3 CP searches for commands stored in these packages and selects routines contained therein for execution. The resident commands inside the ZCPR3 CP are also in the form of a command package, and they are resolved in the same way.

Input/Output Packages

Input/Output Packages (IOPs) execute in conjunction with the low-level, character-oriented input/output routines in the BIOS. The BIOS simply directs its calls for console, list, punch, and reader input/output to the routines in the current IOP for execution.

CP/M Redirectable Input/Output

Redirectable Input/Output refers to the ability of the user to switch between various Input/Output devices during the course of a session.

CP/M supports four logical I/O devices: a console, a printer, a reader, and a punch. Each of these four logical devices may have any one of four physical devices assigned to it, giving the user up to sixteen physical devices. The I/O Byte (at memory location 3) is used to specify the assignment of physical to logical devices. For further details, refer to the discussion of I/O in Chapter 2.

By using the STAT command, the CP/M user may change these assignments from time to time without actually changing the hardware configuration of his system. For instance, my system comes up with the CRT as the principal console device. If I so wish, however, I can assign my printing terminal as the console by issuing the command "STAT CON:=TTY:"; I then enter all my commands at the printing terminal keyboard and, when finished, move back to the CRT by issuing the command "STAT CON:=CRT:".

ZCPR3 Redirectable I/O System

Under ZCPR3, a slightly different scheme for redirectable I/O has been implemented. However, the implementer has the choice of continuing to use the CP/M scheme or of switching to this new one.

When a ZCPR3 system cold boots, the BIOS, as loaded from the system tracks of the disk, contains only a few primitive I/O drivers. The CRT physical device is assigned to the console, reader, punch, and list logical devices; thus all output goes to the CRT. No redirection is permitted at this time.

The BIOS is structured so that all the I/O entries in its jump table branch to corresponding entries in a second jump table (which is initialized by the Cold Boot Routine). This second jump table is placed on a page boundary at the beginning of a scratch area. It is in this scratch area that the physical device drivers for the system reside. It is recommended that this scratch area be approximately 2K in size, to permit maximum flexibility. Figure 10-1 shows this new BIOS layout in memory.

What the Redirectable I/O System Buys You

As the reader can see, CP/M compatibility is maintained in that there are still only the four standard logical devices: Console (CON:), Printer (LST:), Reader (RDR:), and Punch (PUN:). What the reader may not see yet is that the restriction of assigning only four physical devices to each of these logical devices is now gone, as well as the rather obscure names given by CP/M to the physical devices (such as UR1:, UC1:, BAT:, etc).

The advantage of this system is that it does not restrict the designer to the I/O Byte structure defined by Digital Research. Instead, the designer can structure the I/O Byte any way he chooses or even select a different structure, such as a 2-byte I/O Word, to control his redirectable I/O.

```
Base of BIOS -->     ------------------------------------
(Lower Memory)       | Jump for Cold and Warm Boots      |
                     | Console Status Jump to IOBASE+12  |
                     | Console Input Jump to  IOBASE+15  |
                     | Console Output Jump to IOBASE+18  |
                     | List Output Jump to    IOBASE+21  |
                     | Punch Output Jump to   IOBASE+24  |
                     | Reader Input Jump to   IOBASE+27  |
                     | Jumps for Disk I/O                |
                     | List Status Jump to    IOBASE+30  |
                     | Jump for Sector Translation       |
End of Jump Table ->|-----------------------------------|
                     | Body of BIOS, Containing:         |
                     |    Cold and Warm Boot Routines    |
                     |    Disk I/O Routines              |
                     |    Sector Translation Routine     |
IOBASE ->            |-----------------------------------|
(On Page Boundary,   | Jump to Status Routine            |
2K Bytes in Size)    | Jump to Device Select Routine     |
                     | Jump to Device Name Routine       |
                     | Jump to Package Init Routine      |
IOBASE + 12 -->      | Jump to Console Status            |
IOBASE + 15 -->      | Jump to Console Input             |
IOBASE + 18 -->      | Jump to Console Output            |
IOBASE + 21 -->      | Jump to List Output               |
IOBASE + 24 -->      | Jump to Punch Output              |
IOBASE + 27 -->      | Jump to Reader Input              |
IOBASE + 30 -->      | Jump to List Status               |
IOBASE + 33 -->      | Jump to New I/O Routine           |
End of Jump Table ->|-----------------------------------|
                     | Body of Redirectable I/O Driver   |
                     |  Package Containing the Routines  |
                     |  Jumped to Starting at IOBASE     |
End of BIOS -->      ------------------------------------
```

Figure 10-1. BIOS with Redirectable I/O Package

Environments such as the following can be implemented:
No Reader or Punch Devices are available
The LST: Device may be any of the following:

1) Printing Terminal
2) CRT
3) Line Printer
4) Modem
5) Link to Another Computer
6) Disk File

The CON: Device may be any of the following:

1) Printing Terminal (TTY)
2) CRT
3) Modem
4) Link to Another Computer
5) CRT Input and CRT and Modem Output
6) CRT Input and CRT and Remote Computer Output
7) CRT Input and CRT and Disk File Output
8) TTY Input and TTY and Modem Output
9) TTY Input and TTY and Remote Computer Output
10) TTY Input and TTY and Disk File Output

As the reader can see, this structure supports six LST: devices and ten CON: devices (which is quite a bit more flexible than having only four LST: devices and four CON: devices). Also, six devices require only 3 bits to represent them (0-5) and ten devices require only 4 bits to represent them (0-9), so we still require only one I/O byte for I/O redirection.

The Redirectable I/O Drivers contain three routines (the first three in the jump table starting at IOBASE) which provide the following functions:

1) STATUS Tells the calling program how many physical devices are available for the CON:, LST:, RDR:, and PUN: logical devices and which physical device is currently assigned to each of the logical devices

2) SELECT Allows a calling program to assign a physical device to a logical device.

3) NAMER Returns to the calling program a pointer to a text string which describes a physical device. The calling program passes to this routine the logical device and physical device numbers.

The STATUS routine requires no inputs and returns a pointer to a table in HL. This table is structured as a series of four byte pairs. The first byte pair is associated with the CON: device, the second with the RDR: device, the third with the PUN: device, and the fourth with the LST: device. The first byte of each pair contains the number of physical devices that may be assigned to the associated logical device; (this number is in the range 0 to 255). The second byte of each pair contains the device number of the physical device currently assigned to the logical device (in the range 0 to [number of devices - 1]). Table 10-1, below, shows the structure of the 8-byte Status Routine table and typical assignments in the code.

Table 10-1. Status Routine Table Structure

Device	Count Byte Number	Assignment Byte Number
CON:	0	1
RDR:	2	3
PUN:	4	5
LST:	6	7

```
            Code Example:
STABLE:
     DB   6,2          ;6 Devices, Device 2 (3rd Device)
                       ; Assigned to CON:
     DB   0,0          ;No RDR: Devices
     DB   0,0          ;No PUN: Devices
     DB   10,5         ;10 Devices, Device 5 (6th Device)
                       ; Assigned to LST:
```

The SELECT routine assigns a physical device to a logical device. It is called with the logical device number (where CON: is 0, RDR: is 1, PUN: is 2, and LST: is 3) in the B Register and the physical device number in the C register. SELECT returns with the Zero Flag Set (Z) if an invalid selection was made (such as B > 3 or C > max device number). Example:

```
MVI  B,0       ;Select CON:
MVI  C,4       ;Physical Device 4
CALL IOBASE+3  ;SELECT Routine
JZ   DEVERR    ;Error Handler
MVI  B,3       ;Select LST:
MVI  C,2       ;Physical Device 2
CALL IOBASE+3  ;SELECT Routine
JZ   DEVERR    ;Error Handler
```

Finally, the NAMER routine returns to the caller a text string (a vector of ASCII characters terminated by a binary 0) that describes the physical device requested. On input, B contains the logical device number and C contains the physical device number (as in the SELECT routine). On output, HL points to the string and the Zero Flag is Set (Z) if an invalid selection was made. This returned text string contains the mnemonic name of the device (up to eight characters long) followed by a space and any desired text which describes the attributes of the physical device. Example:

```
MVI  B,0       ;Select CON:
```

```
MVI  C,2        ;Physical Device 2
CALL IOBASE+6   ;NAMER Routine
JZ   DEVERR     ;Error Handler
CALL PSTR       ;Print String pted to by HL
```

may result in the following text being printed:

```
MODEM DC Hayes Smartmodem
```

Note that this conforms to entry 3 (Physical Device 2 is the third device) in the table a couple of pages back.

To put this all together, the tools DEV and DEVICE2 are provided with the ZCPR3 System. They perform the following functions:

- o Return the Names of All Physical Devices
- o Return the Names of only the Physical Devices Associated with a Particular Logical Device
- o Allow the User to Assign a Physical Device to a Logical Device

Loading Redirectable I/O Drivers

The Cold Boot Routine in the main body of the BIOS initializes the I/O package starting at IOBASE, providing an initial jump table and an initial set of routines.

Once the Cold Boot Routine has finished, it passes control to ZCPR3. One of the things it has done before it passes control, however, is to store an initial command, STARTUP, in the Multiple Command Line Buffer. ZCPR3 starts up, sees this command in its buffer, and executes it. Upon execution, STARTUP loads the Multiple Command Line Buffer with a series of commands, which include a command to load a *.IOP file.

Once LDR has completed loading the IOP file, it calls the driver initialization routine at IOBASE+9 and returns to ZCPR3 when this routine is finished. The initialization routine performs whatever device initializations are desired.

The source code of the I/O Package named SYSIO.ASM provides a useful example of a Redirectable IOP.

Named Directory Data Files

The *.NDR files contain definitions of ZCPR3 Named Directory environments. These definitions consist of the assignment of mnemonics and passwords to directory references. The *.NDR files are created by the MKDIR command or by assembling the SYSNDR.ASM file. The SYSNDR.ASM file shows the internal structure of a Named Directory Data File, which is:

```
DB   DISK,USER       ; disk A = 1
DB   'NDIRNAME'      ; 8 characters for a name
DB   'PASSWORD'      ; 8 characters for a password
                     ;   space-fill if none
 ...
DB   0               ; end of NDR indicated by a zero
                     ;   in place of a disk number
```

The Environment Descriptor contains the starting address of the memory image for the NDR file and the number of 18-byte entries it contains.

TCAP Data Files

The *.Z3T overlays the second half of the ZCPR3 Environment Descriptor with a data entry extracted from the Z3TCAP.TCP file or with an entry created by the TCMAKE command. The structure of this data file is described elsewhere in this book. The TCAP entry is a part of the ZCPR3 Environment Descriptor; for this reason, the LDR command should load a *.Z3T file *after* it has loaded the *.ENV file. If the *.Z3T file is loaded first, it will be overlaid by the *.ENV file and its effect will be lost.

11 Inside the ZCPR3 Message Buffers

The operation of the ZCPR3 System is dependent upon the ZCPR3 Message Buffers located at the symbol Z3MSG in the Z3BASE.LIB file. The messages are used in so many different contexts by several different types of ZCPR3 tools that the total picture is somewhat difficult to see. For this reason, the following is presented to summarize the contents and application of the ZCPR3 Message Buffers. The following figure summarizes the ZCPR3 Message Buffers:

Offset from Z3MSG	*Function*
0	Error Flag (Error Handler)
1	IF (Current IF Level)
2	IF Active
3	ZCPR3 Command Status
4-5	Error Address (for Error Handler)
6	Program Error Code
7	ZEX Message Byte
8	ZEX Running Flag
9-10	Address of Next ZEX Char
11-12	Address of First ZEX Char
13	Shell Control Byte
14-15	Shell Scratch
10H-2FH	Error Command Line
30H-39H	Registers
3AH-3FH	Reserved
40H-4FH	User-Defined

Error Handler Messages

The *Error Flag* at Z3MSG+0 is used to indicate if an Error Handler is enabled. If this flag is 0, there is no Error Handler; if this flag is not zero (0FFH), there is an Error Handler, and the *Error Command* stored at Z3MSG+10H to Z3MSG+2FH (32 bytes allowed, including terminating zero) is the command line executed when the ZCPR3 CP executes an Error Handler.

Both the Error Flag and Error Command are set by an Error Handler and read by the ZCPR3 CP as required.

IF Messages

The *IF* message at Z3MSG+1 and the *IF Active* message at Z3MSG+2 are used to indicate the number of the current IF level and the Flow Command State at that and all preceding levels. The IF message has either no bit set (which indicates that no IF is active) or one bit set (which indicates which IF level is active). Eight levels of active IFs are permitted. Bit position 0 indicates IF level 1 and bit position 7 indicates IF level 8. The IF Active message indicates the Flow Command State of each of the eight IF levels, and it tracks on a one-to-one relationship with the IF message.

The IF message assigns an IF level to each bit, and only one bit is set to indicate which IF level is currently active:

```
-----------------
| | | | | | | | | IF message (no active IF)
-----------------


-----------------
| | | | | | | |1| IF message (IF level 1 is current)
-----------------


-----------------
|1| | | | | | | | IF message (IF level 8 is current)
-----------------
```

The IF Active message is used to hold the states of all IF levels up to and including the current one:

```
-----------------
| | | | |0|1|1|1| IF Active (IF levels 1-3 TRUE)
-----------------
```

The following example illustrates how the two IF message bytes work together:

```
-----------------
| | | | |1|0|0|0| IF Message (Current Level is 4)
-----------------


-----------------
| | | | |0|0|1|1| IF Active (IF levels 1-2 TRUE)
-----------------
```

Flow Command State

From this example, the reader can see that four IF commands have been issued, and the third IF enabled a FALSE condition, which automatically made the fourth IF FALSE. The current Flow Command State is determined by the following algorithm:

```
if IF_message = 0 then flow_command_state = TRUE
    else flow_command_state = IF_message AND IF_Active_message
```

The two IF messages are usually set by an FCP routine or a transient and they are read by the ZCPR3 CP to determine the Flow Command State.

Command Status

The *Command Status* of the ZCPR3 CP is stored in Z3MSG+3. This byte can take on one of three values:

0	the current command is invoked normally
1	the current command is a Shell
2	the current command is an Error Handler

This message is set by the ZCPR3 CP and read by a Shell or an Error Handler when it is invoked. Each type of tool will either install itself or execute its function, depending on the setting of this message. In the case of a Shell, it will assume the role of the ZCPR3 CP for input processing and clear this message in order to allow it to invoke other Shells under it.

Error Address

If a command is in error, the ZCPR3 CP will set the Command Status message to 2 in order to indicate to the Error Handler that it was invoked as such and it will store the address of the first byte of the command line in the *Error Address* message at Z3MSG+4 and Z3MSG+5 (low order, high order, resp). The Error Handler may then use this pointer to examine the command in error.

Program Error Code

A byte is reserved at Z3MSG+6 to contain a *Program Error Code.* This byte is set to 0 to mean that no error condition exists or non-zero (any desired value) to indicate that an error condition exists and what that condition is. The exact definition of these values depends upon the tool which sets them. The only convention to be observed is that no error is indicated by a zero value and an error is indicated by a non-zero value.

The IF condition called ERROR examines the Program Error Code and sets the next Flow Command State according to its value. In essence, a tool can set the Program Error Code and pass this information on to a tool which is later executed. The ZCPR3 CP itself ignores the Program Error Code—the code is used only to pass information from one tool to another.

ZEX Message

The *ZEX Message* byte is used to pass control information to the ZEX Monitor if it is in operation. This byte can have one of three values:

0	normal operation
1	a ZCPR3 prompt has appeared
2	suspend ZEX Monitor activity

During normal operation, the ZEX Monitor provides input characters whenever a BIOS console input call is made. The ZCPR3 prompt message is used to tell ZEX that command line input is being requested and that the ZEX Monitor can activate to provide this input. The command to suspend ZEX Monitor activity simply tells ZEX to stop providing input characters and allow input to come from the console. This state continues until the ZEX Message byte is set to some other value.

The ZEX Message byte is set by the ZCPR3 CP when it prompts for input from the console and after this input is accepted. The ZEX Message byte should be set by any ZCPR3 Shell as it would be by the ZCPR3 CP, and it may be set by any other tool.

ZEX Running Flag

The message at Z3MSG+8 is the *ZEX Running Flag*, and it is set by ZEX to a non-zero value to indicate that the ZEX Monitor is in operation. When the ZEX Monitor terminates, it resets this message to zero.

ZEX Control

The message at Z3MSG+9 and Z3MSG+10 is the address of the next character in the ZEX memory-based file image which will be returned by ZEX on the next BIOS console input call. The message as Z3MSG+11 and Z3MSG+12 is the address of the first character of the ZEX memory-based file image, where the file image is stored in sequence from high memory to low memory.

These two messages are used to control the execution flow of commands under ZEX. The GOTO command, for example, will search the ZEX memory-based file image for the label referenced in the command starting at the address of the first character (in Z3MSG+11 and Z3MSG+12) and, if found, will set the address of the next character for ZEX processing in the message at Z3MSG+9 and Z3MSG+10. The end of the ZEX memory-based file image is flagged by a byte of value 0FFH.

Shell Control Byte

The *Shell Control Byte* is located at Z3MSG+13, and it is set by Shells and other tools to control the features of the Shell execution. The following convention has been adopted for the usage of this message:

Bit 0	set Shell to comment mode
Bit 1	have Shell echo the command lines it builds
Bit 7	have Shell wait upon entry before execution

The use of other bits is open to the discretion of the designer.

Shell Scratch

The *Shell Scratch* messages are at Z3MSG+14 and Z3MSG+15. Their use is undefined and left open to the Shell designer.

Registers

The *Registers* are located from Z3MSG+30H to Z3MSG+39H. These are 10 one-byte registers which may be used to store any values desired. The REG tool can directly set their values, and the IF tools can test them.

Reserved Messages

The messages from Z3MSG+3AH to Z3MSG+3FH are reserved for the ZCPR3 System designer (Richard Conn) and should not be used by any other person. This will leave an opening for future expansion of the ZCPR3 System.

User-Defined Messages

The messages from Z3MSG+40H to Z3MSG+4FH are available for user definition and may be used for any purpose desired by the software designer.

12 Inside ZEX

ZEX Command File Processor

The ZEX Command File Processor is an integral part of the ZCPR3 System—much more so than any other tool. It reacts closely with the ZCPR3 Command Processor, and its execution can be controlled and directed by any ZCPR3 Tool. Refer to the section on ZEX in Chapter 3 for a user's viewpoint of its operation.

When the ZEX Command File Processor executes, it places the ZEX Monitor in high memory, just under the ZCPR3 CP. Beneath the ZEX Monitor is the text of the command file it is to execute; this text has been preprocessed by the ZEX Command File Processor for greater ease of use and efficiency of execution by the ZEX Monitor. After installation of both the ZEX Monitor and memory-resident command file is complete, the BDOS entry point address is adjusted to protect the monitor and its command file and execution resumes with the ZEX Monitor providing input from the command file. Figure 12-1 shows the picture of the ZCPR3 System.

```
High Memory -> ---------------------------------
               | Extended Buffers and Packages |
               ---------------------------------
               | BIOS                          |
               ---------------------------------
               | BDOS                          |
Normal Ptr  -> ---------------------------------
  to BDOS      | ZCPR3 Command Processor       |
               ---------------------------------
               | ZEX Monitor                   |
               ---------------------------------
               | Command File                  |
               ---------------------------------
               | ZEX Monitor Front-End         |
Pointer to  -> ---------------------------------
      BDOS     | Transient Program Area        |
               | (TPA)                         |
               ---------------------------------
               | Buffers                       |
Low Memory  -> ---------------------------------
```

Figure 12-1. ZCPR3 System with ZEX Monitor Installed

The command file is stored from higher memory down, and its termination is signalled by a 0FFH byte. The ZEX Monitor Front-End consists of mainly a JMP BDOS_ENTRY instruction.

During the execution of the ZEX Monitor, several messages in the ZCPR3 Message Buffers enable the ZCPR3 CP and the ZCPR3 Tools to interact with the ZEX

Monitor. Operations with the ZEX Monitor, such as suspending its operation, enabling its operation, informing it that a command prompt has appeared, and changing its execution flow within the command file are possible by means of these messages. The messages which affect the ZEX Monitor are summarized in the following table:

Offset from Z3MSG	*Function*
7	ZEX Message Byte
8	ZEX Running Flag
9-10	Address of Next ZEX Char
11-12	Address of First ZEX Char

ZEX Message

The ZEX Message byte is used to pass control information to the ZEX Monitor if it is in operation. This byte can have one of three values:

0	normal operation
1	a ZCPR3 prompt has appeared
2	suspend ZEX Monitor activity

The ZEX Monitor provides input characters whenever a BIOS console input call is made. The ZCPR3 prompt message is used to tell ZEX that command line input is being requested. The ZEX Monitor will leave a suspended state if it is in one at this point and provide input to the command line with whatever special command line processing is needed to be done. The command to suspend ZEX Monitor activity tells ZEX to stop providing input characters and allow input to come from the console. This state continues until the ZEX Message byte is set to some other value.

The ZEX Message byte is set by the ZCPR3 CP when it prompts for input from the console and after this input is accepted. The ZEX Message byte should be set by any ZCPR3 Shell as it would be by the ZCPR3 CP, and it may be set by any other tool.

ZEX Running Flag

The message at Z3MSG+8 is the ZEX Running Flag, and it is set by ZEX to a non-zero value to indicate that the ZEX Monitor is in operation. When the ZEX Monitor terminates, it resets this message to zero.

ZEX Control Messages

The message at Z3MSG+9 and Z3MSG+10 is the address of the next character in the ZEX memory-based file image which will be returned by ZEX on the next BIOS console input call. The message as Z3MSG+11 and Z3MSG+12 is the address of the first character of the ZEX memory-based file image, where the file image is stored in sequence from high memory to low memory.

These two messages are used to control the execution flow of commands under ZEX. The GOTO command, for example, will search the ZEX memory-based file image for the label referenced in the command starting at the address of the first character (in Z3MSG+11 and Z3MSG+12) and, if found, will set the address of the next character for ZEX processing in the message at Z3MSG+9 and Z3MSG+10. The end of the ZEX memory-based file image is flagged by a byte of value 0FFH.

13 Inside the ZCPR3 Shells

Shell

A *Shell* is a special type of tool under ZCPR3. It should generally conform to the rules of a ZCPR3 tool (see the chapter on ZCPR3 Tools). A Shell performs one of two basic functions when it executes: (1) it installs itself or (2) it performs its shell-oriented function, accepting commands from the user and generating command lines for the ZCPR3 CP to execute.

Shell Self-Installation

When a Shell installs itself, its primary mission is to push a command line onto the Shell Stack. The ZCPR3 Environment Descriptor contains a pointer to the Shell Stack; if there is no Shell Stack available (the value of the pointer is zero), then the Shell cannot function and should so indicate.

In pushing its command line onto the Shell Stack, a Shell can determine its name from the External FCB. It should also initialize any message buffers it wants to use to communicate with itself between invocations. There are several sets of buffers available with which to do this.

At location Z3MSG+13 is a Shell Control Byte. It is structured as follows:

Bit 0	Enable Shell Comment
Bit 1	Enable Shell Echo
Bits 2-6	Use as Desired
Bit 7	Enable Shell Entry Wait

It is recommended that implementations of Shells adhere to the indicated usage of these bits. Their meanings are described below.

Enable Shell Comment. The Shell should run in a comment mode when this bit is set (if appropriate). All inputs should be flushed as comments rather than handed to the ZCPR3 CP for processing unless preceded by a command escape character (which is recommended to be the exclamation mark). Refer to the description of the Shell named SH for an example.

Enable Shell Echo. The Shell should print the command line it builds to pass to the ZCPR3 CP if this bit is set.

Enable Shell Entry Wait. If this bit is set when the Shell is invoked by the ZCPR3 CP, the Shell should print a prompt to the user and wait for his response before resuming execution. The purpose of this feature is to give the user time to look at the results of the last command executed before the Shell comes into play, possibly wiping out the previous display (as when the Shell is screen-oriented).

Locations Z3MSG+14 and Z3MSG+15 are reserved for use by Shells. This usage is undefined, but one intended application is to make a note of the disk and user area the user was in at the time the Shell exited to run a command line.

Other useful buffers for Shell-oriented messages are the registers at Z3MSG+30H to Z3MSG+39H and the System-Wide File Names in the Environment Descriptor.

Some useful Shell-oriented Z3LIB routines are:

GETSH2	determine if a Shell Stack is available
QSHELL	determine if the process was invoked as a Shell
PUTSHM	put values to Shell messages at Z3MSG+13 to Z3MSG+15

GETSHM	get values to Shell messages at Z3MSG+13 to Z3MSG+15
GETFN1	get pointer to System-Wide File Names
SHPUSH	push command line onto Shell Stack
SHPOP	pop top element off of Shell Stack

Shell Execution

When a Shell executes, it should have two basic operating modes: (1) as a ZEX command line-forwarder and (2) as a user command generator.

If the Shell is not command-line oriented (like VFILER), then it should test to see if ZEX is running, and, if so, accept the next command line from ZEX and pass it through directly to the ZCPR3 CP. A simple way to do this is (1) to set the ZEX message (at Z3MSG+7) to 1 indicating a prompt, (2) input the line via a BDOS call, (3) clear the ZEX prompt message to 0, and (4) clear the Command Status message (at Z3MSG+3) to zero. In this way, if a ZEX command file is in execution from a Shell-invoked command, then the command file can continue to completion. If a Shell is command-line oriented (like SH), no special attention need to be paid to the input source other than accepting normal BIOS input and setting the ZEX message byte to indicate a prompt.

A Shell should act like the ZCPR3 CP if it inputs a command line from the user. In particular, it should set the ZEX message byte at Z3MSG+7 to 1 to indicate that a prompt has appeared before it inputs its line and it should reset this byte back to zero after the line is accepted. All ZCPR3 Shells should clear the Command Status message to zero before they return to the ZCPR3 CP to run their commands so that one Shell can successfully invoke another Shell.

Some useful Z3LIB routines are:

GETSHM	get Shell message
PUTSHM	put Shell message
PUTZEX	put ZEX message byte
PUTCST	put ZCPR3 Command Status message
GETCST	get ZCPR3 Command Status message
PUTCL	store command line in buffer

14 Inside the ZCPR3 Error Handlers

An *Error Handler* is a special type of a ZCPR3 Tool which is invoked in two different situations: (1) by the user when he wants to install it and (2) by the ZCPR3 Command Processor when an error occurs. As a ZCPR3 Tool, an Error Handler should observe the conventions of a ZCPR3 Tool; see the chapter on the internals of ZCPR3 Tools for details.

Whenever the ZCPR3 CP runs a command, it leaves a Command Status message at Z3MSG+3 (the third byte from the beginning of the ZCPR3 Message Buffer, where the first byte in the buffer is numbered 0). If this message has the value of 0, then the command was issued by the user. If it has a value of 1, then the command was issued by the ZCPR3 CP as a shell invocation. If Z3MSG+3 has a value of 2, then the command was issued by the ZCPR3 CP as an error handler. Shortly after an Error Handler begins running, it should check this message to see how it was invoked. If it was not invoked as an error handler (i.e., the user issued the command), then its function is to install itself as an error handler or print a help message if the // option is presented. If the Error Handler was invoked as an error handler by the ZCPR3 CP ((Z3MSG+3) = 2), then it should process the command in error as indicated by the ZCPR3 CP.

The QERROR routine in Z3LIB is useful for performing a check on Z3MSG+3 and seeing if an Error Handler was invoked.

Error Handler Self-Installation

An Error Handler installs itself by storing a command line to invoke itself in the Error Command message at Z3MSG+10H to Z3MSG+2FH and setting the error code at Z3MSG+0 to 0FFH (non-zero). It is best to have an Error Handler obtain its name from the External FCB so that the user can name an error handler whatever he wishes.

The algorithm for self-installation of an Error Handler is:

```
if external_FCB_exists then
     error_handler_name = external_FCB_name
else
     error_handler_name = default_name
end if
error_command_buffer = error_handler_name
error_code = 0FFH
```

Useful Z3LIB routines for these functions are:

GETEFCB	determine existence of External FCB and set pointer to it if it exists
PUTERC	store error command line into buffer
PUTER1	set error code message

Error Handler Execution

When an Error Handler executes, it must determine the location of the command in error in order to perform any processing on this command. The ZCPR3 CP provides this information in the form of an address at Z3MSG+4 and Z3MSG+5 (low order and high order address, resp). Once this address is known, the Error Handler can examine the command. Note that this command is terminated either by a binary zero (0) or a semicolon (';').

Useful Z3LIB routines are:

ERRADR	get address of command in error
GETCLn	get data on the command line buffer

15 Inside the ZCPR3 Tools

Every ZCPR3 tool (COM file) for a particular ZCPR3 System installation has the same basic internal structure. There are two basic types of tools: (1) those with external Environment Descriptors and (2) those with internal Environment Descriptors. The ZCPR3 System tools are distributed with external Environment Descriptors, and this is the recommended configuration.

If a ZCPR3 tool uses an external Environment Descriptor, its internal structure is as follows:

```
100H:
     JMP   START_OF_TOOL             ; JMP to executable code
     DB    'Z3ENV'                   ; Environment Descriptor indic
     DB    1                         ; Env Desc is External
Z3EADR:
     DW    Z3ENV                     ; Address of Env Desc
     ...
START_OF_TOOL:
     ...                             ; executable code
```

The Z3INS (ZCPR3 Installation) tool installs a ZCPR3 tool which uses an external Environment Descriptor by setting the pointer to the Environment Descriptor at Z3EADR. Z3INS obtains the address of the external Environment Descriptor from the SYS.ENV file that is specified to it at execution time.

If a ZCPR3 tool uses an internal Environment Descriptor, then its internal structure is as follows:

```
100H:
Z3EADR:
     JMP    START_OF_TOOL
     DB     'Z3ENV'
     DB     0                      ; Internal Env Desc

< Environment Descriptor >

     ...
START_OF_TOOL:
     ...                           ; executable code of tool
```

The body of the tool can contain whatever code is necessary to perform the function desired. In order to fall in line with the philosophy of the ZCPR3 System, it is recommended that all ZCPR3 Tools observe the following conventions:

1. The command invocation form of a tool is:

   verb arg1 arg2 ... option

If any argument is optional, then the form:

```
verb arg1 ... /option
```

should be recognized to permit options when a command supports a variable number of arguments.

2. Built-in documentation on the tool should be invoked by a command of the form:

```
verb //
```

This built-in documentation should take the format of:

```
VERB version_number
Syntax:
     text_describing_syntax_of_use
other_text_as_desired
```

3. If it is within reason for a tool to accept a list of files as an argument, then this list should consist of file references separated by commas, like:

```
verb file1,file2,... filea,fileb,...
```

The above format contains two lists of files as arguments.

4. If a directory reference is to be allowed, both the DU and DIR forms should be permitted.

The library Z3LIB contains a large number of routines which make meeting the above requirements quite simple. Various parsers and other routines are included which reduce these requirements to simple subroutine calls.

Section 3

Installation

This section of the book is devoted to the topic of installing a ZCPR3 System, which includes the ZCPR3 Command Processor, the System Segments, and the ZCPR3 Toolset. Knowledge of assembly language and the basic programming concepts of CP/M is required to follow this section of the book.

16 Overview of ZCPR3 Installation

Introduction

Installation of ZCPR3 (unless you have the commercial product, Z3-DOT-COM) is an involved process, and the installer must have a working knowledge of the following:

1. 8080 and Z80 assembly language programming
2. CP/M 2.2
3. the CP/M SYSGEN procedure

Fortunately, Echelon, Inc. supplies a version of ZCPR3 called Z3-DOT-COM that is extremely simple to install on your system—a submit file does nine-tenths of the work for you, and while this is running, the installation program reports successful installation of all the major command processors, buffers, and utilities. Once the main installation has completed, a simple command causes the installation program to install the extended command processor ZEX, using the environment descriptor generated by the main installation pass. These procedures can be performed even by a person with minimal knowledge of CP/M and assembly language programming.

Nevertheless, to get full use of Z3-DOT-COM, and to adapt it to your own needs and preferences, you will require information provided in this section. It is therefore strongly recommended that, when you have become familiar with the operation and major features of Z3-DOT-COM, you carefully read this section so that you will be able to install new features and change existing tools to suit your preferred working habits.

Three parts of the system must be created or initialized during the installation process:

1. the *Operating System* or *SYSGEN image*, which is present on the system tracks for most computers and includes a disk boot, the ZCPR3 Command Processor, the CP/M 2.2 BDOS or ZRDOS, and a modified BIOS.

2. the ZCPR3 *System Segments*, which are independent files that may be loaded from disk into the appropriate places in memory by the ZCPR3 utility named LDR.COM

3. the various ZCPR3 *utilities*, each of which has to be provided with the address of the ZCPR3 Environment Descriptor

The ZCPR3 System is tied together by the ZCPR3 Environment Descriptor, which is a set of buffers that passes information between all elements of a ZCPR3 System.

The ZCPR3 Environment Descriptor contains information such as the addresses of the System Segments, the addresses of several buffers which are significant to ZCPR3, data on what ZCPR3 resources are available, and information about the physical attributes of some of the input/output devices connected to the system (such as the number of columns and lines on the CRT screen).

Operating System Memory Images

The installer must build a proper SYSGEN Memory Image of the target ZCPR3 System (the *target operating system* is the system being built, as opposed to the *host operating system* which is the system used to build the target system). In building the target system, the ZCPR3 Command Processor must be assembled and a BIOS containing a modified Cold Boot routine must be prepared.

System Segments

The installer must select and assemble the various ZCPR3 System Segments to be used in conjunction with the target ZCPR3 System. A *System Segment* is a file that is loaded into a fixed location in memory by the LDR.COM utility. Each System Segment remains memory-resident until a new System Segment is loaded over it. Depending on the commands issued, the ZCPR3 Command Processor or a ZCPR3 utility may call upon a loaded System Segment to perform a function or provide information.

All System Segments must be initialized by the Cold Boot routine in the BIOS of the target ZCPR3 System. This initialization consists of zeroing out the first N bytes of each segment's memory buffer, where N depends upon the segment being initialized.

The following are the System Segments which are supported by ZCPR3. Each System Segment has a distinctive file type, and LDR.COM recognizes this and loads each segment differently.

Segment File Type	*Function of System Segment*
*.ENV	Environment Descriptor, including a TCAP
*.Z3T	ZCPR3 TCAP Entry
*.FCP	Flow Command Package
*.IOP	Input/Output Package
*.NDR	Named Directory File
*.RCP	Resident Command Package

A *package,* as referred to above, is a set of executable subroutines which is divided into two parts: the visible section, through which an interface to the routines is provided, and the hidden section, which contains the code of the routines. As a System Segment, a package can be loaded dynamically any time during a terminal session by running the LDR.COM utility.

Flow Command Packages. A *Flow Command Package* is a package which implements the ZCPR3 flow commands. These commands are IF, ELSE, FI (same as ENDIF), and XIF (exit all IFs), and their function is to control the flow of command execution by setting the *Flow State* to TRUE or FALSE. If the Flow State is TRUE, all commands are allowed to execute; if the Flow State is FALSE, only Flow Commands (IF, ELSE, FI, and XIF) are allowed to run.

An example of a command sequence containing flow commands is:

```
IF EXIST MYFILE.TXT
     TYPE MYFILE.TXT
ELSE
     ECHO MYFILE.TXT DOES NOT EXIST
FI
```

Input/Output Packages. An *Input/Output Package* is a package which contains a set of input/output drivers. An I/O Package provides the low-level device drivers, referenced by the BIOS, that support console input/output, list output, punch output, and reader input. Any I/O Package can support many more console, list, punch, and reader devices than the standard CP/M I/O byte; further, the package loaded at start-up can be replaced with a different package that supports a different set of physical devices, merely by running the LDR.COM utility.

Resident Command Packages. A *Resident Command Package* is a collection of memory-resident commands that supplement the commands resident within the ZCPR3 Command Processor itself (the *ZCPR3-Resident Commands*). These commands replace a number of COM files by one *.RCP file; because they are memory-resident, they execute very quickly without any need for additional disk accesses. When the user issues a command, the current RCP is checked for a match of the command before a disk access is performed to search for a matching COM file. For a full description of the process, refer to the section on "Command Search Hierarchy" in Chapter 2.

Environment Descriptor. The *ZCPR3 Environment Descriptor* is a data file that contains information on several attributes of the ZCPR3 System. Additionally, the Environment Descriptor contains a *ZCPR3 TCAP (Terminal Capabilities)* entry that describes various attributes of the console CRT, such as the sequence of characters to cause its screen to clear or to position its cursor.

ZCPR3 Named Directories. The *ZCPR3 Named Directory* file contains data relating a mnemonic, such as PASCAL or ROBERT, with a Disk and User Area (a logical directory). Under ZCPR3, either Disk/User (DU) forms or Named Directories can be used to refer to logical directories:

```
DIR A15: DIR ROOT:
```

Utilities

To be used effectively as a part of a ZCPR3 System, all ZCPR3 utilities must be initialized to contain either (1) a pointer to the ZCPR3 Environment Descriptor (if such a descriptor is available as a System Segment) or (2) the ZCPR3 Environment Descriptor itself.

The ZCPR3 utility Z3INS.COM performs this initialization. Z3INS.COM will install a group of utilities with the required information very quickly and make this process relatively painless. Z3INS.COM itself does not need to be installed but can be for the sake of consistency.

Other Basic Concepts

For a general description of ZCPR3 basic concepts, refer to Chapter 2. In this section, we give only technical information required for installation or modification of the system.

Command Search Path. The *Command Search Path* is a buffer which contains an expression (in the form of byte pairs) of the sequence of directories to examine when the ZCPR3 Command Processor searches for a COM file. It is recommended that this buffer be placed outside the ZCPR3 Command Processor (be enabled as an *External Path*) so that the ZCPR3 utilities may readily access and modify it.

The elements of a Command Search Path are byte pairs. The first byte indicates the disk, and the second byte indicates the user area, on which to search. The value of the first byte may be in the range 1-16 to indicate disks 'A' to 'P', or the character '$' to indicate the current disk. The value of the second byte may be in the range 0-31 to indicate user areas 0 to 31, or the character '$' to indicate the current user area. *Current Disk* and *Current User Area* refer to the disk and user area logged in at the time the command was executed by the ZCPR3 Command Processor. A value of 0 in the first byte of a byte pair indicates the end of the Command Search Path.

The following is a sample Command Search Path expression:

```
DB   '$',0      ; Current disk, user area 0
DB   1,'$'      ; Disk A, current user area
DB   1,15       ; Disk A, User Area 15
DB   0          ; End of Path
```

SYSGEN Memory Images

The SYSGEN memory images of a conventional CP/M system and a ZCPR3-based system are presented in Figure 16-1. The actual addresses of the various components may vary from system to system, and the installer should make a note of the starting address of each component in the target system.

```
  Address           CP/M Image               ZCPR3 Image
                 ---------------------    ---------------------
                 | BIOS              |    | BIOS with Modified |
                 |                   |    |   Cold Boot *      |
BDOS+0E00H-->    ---------------------    ---------------------
                 | BDOS              |    | BDOS (No Change)   |
CCP +0800H-->    ---------------------    ---------------------
                 | CP/M 2.2 CCP      |    | ZCPR3 *            |
BOOT+0080H-->    ---------------------    ---------------------
                 | BOOT              |    | BOOT               |
BASE+xxxxH-->    ---------------------    ---------------------
                 | Dead Space/SYSGEN |    | Dead  Space/SYSGEN |
BASE= 100H-->    ---------------------    ---------------------
```

Figure 16-1. CP/M and ZCPR3-based SYSGEN Memory Images

Installation requires a modified BIOS image and a ZCPR3 image to be placed over the original CP/M 2.2 BIOS and CCP images. The rest of the system can stay the same. Those new images are marked with an asterisk (*) in figure 16-1. Typical address values are indicated below:

Value	SYSGEN Image Conventional CP/M	SYSGEN Image Morrow CP/M
xxxxH	800H	~ 980H
BOOT = BASE + xxxxH	900H	1080H
ZCPR3 = BOOT + 80H	980H	1100H
BDOS = ZCPR3 + 800H	1180H	1900H
BIOS = BDOS + 0E00H	1F80H	2700H
End of Operating System	????H	2DFFH

System Segments

Installation of the ZCPR3 System Segments involves selecting the features of the segments and then assembling each segment in turn. It is recommended that the MAC assembler of Digital Research or ZAS assembler of Echelon be used to perform these assemblies.

The ZCPR3 Environment Descriptor (*.ENV file) is created by assembling the file SYSENV.ASM. During this process, the files Z3BASE.LIB and SYSENV.LIB are read in and used by the assembler. Z3BASE.LIB defines the memory configuration of the system and makes up most of the environment descriptor information. SYSENV.LIB contains additional details on the system.

The ZCPR3 TCAP files (*.Z3T) are created by running the TCSELECT or TCMAKE programs. TCSELECT allows the user to select his terminal from a list of predefined terminals, while TCMAKE allows the user to define the attributes of his terminal directly. TCMAKE is for users whose terminal does not appear in the standard Z3TCAP.TCP file.

Flow Command Packages (*.FCP) are created by assembling SYSFCP.ASM. During this process, the files Z3BASE.LIB and SYSFCP.LIB are read in and used by the assembler. SYSFCP.LIB defines the features supported by the Flow Command Package being created.

Input/Output Packages (*.IOP) are created by assembling SYSIOP.ASM. During this process, the file Z3BASE.LIB is read in and used by the assembler. All features of the I/O Package are hard-coded into the source of the package.

Resident Command Packages (*.RCP) are created by assembling SYSRCP.ASM. During this process, the files Z3BASE.LIB and SYSRCP.LIB are read in and used by the assembler. SYSRCP.LIB defines the features supported by the Resident Command Package being created.

Named Directory Files (*.NDR) are created in one of two ways: (1) by assembling the file SYSNDR.ASM or (2) by running the MKDIR.COM ZCPR3 utility. MKDIR.COM allows the user to dynamically edit and create new named directory structures while online.

Utilities

The installation of most of the ZCPR3 utilities involves setting up a file containing the names of the utilities to be installed and running the Z3INS.COM ZCPR3 utility on an Environment Descriptor and this file. Z3INS will install each utility named in the file with the information it needs from the Environment Descriptor.

All ZCPR3 utilities may be installed in this way. In an earlier release of ZCPR3, ZEX 3.0 could not, but version 3.1 of ZEX can, so all ZCPR3 tools are now installed via Z3INS.

Installation Steps

The installation process for ZCPR3 involves these steps:

1. Selecting the features desired for the target ZCPR3 System
2. Planning the memory structure of the target ZCPR3 System (the file Z3BASE.LIB is created)
3. Modifying the Cold Boot routine in the BIOS of the target ZCPR3 System to initialize the selected features which require initialization
4. Enabling the desired features in the ZCPR3 Command Processor (the file Z3HDR.LIB is created)
5. Overlaying the CCP with ZCPR3 and the old BIOS with the new BIOS in the

SYSGEN Image
6. Placing the new SYSGEN Image onto the Operating System tracks of the disk
7. Selecting the options for the desired System Segments and creating the System Segments
8. Installing the desired ZCPR3 utilities

Required Hardware

Hardware Required for Installation. The hardware requirements for the installation of ZCPR3 are as follows:

CP/M 2.2 - based system (or ZCPR3 - based system)
8080 or Z80 microprocessor
32K bytes of memory
110K bytes of disk space for source, BAK, and HEX files
computer terminal

Hardware Required for Running ZCPR3. The hardware requirements for running ZCPR3 are:

ZCPR3 - based system
Z80 microprocessor
48K bytes of memory
110K bytes per disk (recommended minimum)
computer terminal

Components of an Operational ZCPR3 System

Figure 16-2 shows an operational ZCPR3 System. The memory image, system segments, and utilities are briefly discussed from an installation viewpoint in the remainder of this chapter.

Memory Image. The memory image in figure 16-2 shows the memory structure of a ZCPR3 System which includes all of the major features.

Notes:

1. All Areas Above E400H are initialized by the Cold Boot Routine in the BIOS
2. Those Areas marked with (S) are ZCPR3 System Segments

System Segments

Z3BASE1.LIB Z3BASE2.LIB Z3HDR1.LIB Z3HDR2.LIB	The System Segments used in this system are provided in the distribution files of ZCPR3. The ZCPR3 System shown here is defined by the file Z3BASE1.LIB, and a much smaller system which does not include the Resident Command Package, Input/Output Package, and Flow Command Package features (only 1K of additional overhead) is defined in Z3BASE2.LIB. Associated with each of the two Z3BASEn.LIB files is a Z3HDRn.LIB file which defines the features of the ZCPR3 Command Processor.

Figure 16-2. ZCPR3 System Memory Image (Z3BASE1.LIB)

```
      Address
FFFF ------------------------------------------
     | ROM Area (System Dependent)            | 2K
F800 ------------------------------------------
     | ZCPR3 External Stack                   |\
F7D0 ------------------------------------------ \
     | ZCPR3 Command Line Buffer              |  \
F700 ------------------------------------------   \
     | ZCPR3 Memory-Based Named Directory (S) |    |
F600 ------------------------------------------    |
     | ZCPR3 External File Control Block      |
F5D0 ------------------------------------------   1K
     | ZCPR3 Message Buffers                  |
F580 ------------------------------------------    |
     | ZCPR3 Shell Stack                      |    |
F500 ------------------------------------------   /
     | ZCPR3                 | Z3TCAP (S)     |  /
F480 |     Environment       ---------------- /
     |          Descriptor (S)                |/
F400 ------------------------------------------
     | ZCPR3 Flow Command Package (S)         | 0.5K
F200 ------------------------------------------
     | ZCPR3 Input/Output Package (S)         | 1.5K
EC00 ------------------------------------------
     | ZCPR3 Resident Command Package (S)     | 2K
E400 ------------------------------------------
     | ZCPR3 BIOS with Modified Cold Boot     |
     | Routine to Initialize All Elements     | 3.5K
     | of the ZCPR3 System Above              |
D600 ------------------------------------------
     | CP/M BDOS                              | 3.5K
C800 ------------------------------------------
     | ZCPR3 Command Processor                | 2K
C000 ------------------------------------------
     | Transient                              |
     | Program                                | ~48K
     | Area                                   |
 100 ------------------------------------------
     | CP/M and ZCPR3 Buffers                 |256 bytes
   0 ------------------------------------------
```

System Segments

SYSENV.ASM SYSENV.LIB	The Environment Descriptor is created by assembling SYSENV.ASM, which uses Z3BASE1.LIB (renamed to Z3BASE.LIB) and SYSENV.LIB during the assembly process.
SYSFCP.ASM SYSFCP1.LIB SYSFCP2.LIB	Two Flow Command Packages are used in conjunction with this system; they are defined by the files SYSFCP1.LIB and SYSFCP2.LIB. SYSFCP1.LIB defines an FCP which is self-contained and executes without using any external files. SYSFCP2.LIB executes the ELSE/FI/XIF commands within itself, but it executes IF by loading the file IF.COM from the ROOT directory and transferring control to it. This eliminates the restriction of capabilities of the IF command which is imposed by the small size of the FCP.
SYSIOP.ASM	The Input/Output Package used in conjunction with this system is contained in the file SYSIOP.ASM.
SYSRCP.ASM SYSRCP1.LIB SYSRCP2.LIB SYSRCP3.LIB SYSRCP4.LIB	Four RCPs are used in conjunction with this system; they are defined by the four SYSRCPn.LIB files (n is between 1 and 4). Each RCP contains a different set of commands with a different set of options enabled for the included commands.

Utilities. Over 70 utilities are associated with the ZCPR3 System. Each utility uses features of the system, including named directory references, access to the various system segments, access to the TCAP facility, and access to all of the data elements in the ZCPR3 Environment Descriptor that it needs. The ZCPR3 Environment Descriptor is the single source for all information needed by a ZCPR3 utility about the system in which it is running.

Consequently, all ZCPR3 utilities access the ZCPR3 Environment Descriptor in one of two ways: (1) they contain a pointer to the descriptor or (2) they contain the descriptor itself. The Z3INS.COM utility is used to install the ZCPR3 utilities with the address of the Environment Descriptor or the descriptor itself. Class 1 utilities are those who contain a pointer to an environment descriptor, and Class 2 utilities contain the descriptor itself.

Supporting the Environment Descriptor in a global memory buffer is the recommended way to implement a ZCPR3 System. This buys the system two distinct advantages:

1. Each utility needs only 2 additional bytes of overhead (the pointer to the Environment Descriptor) rather than the descriptor itself (which occupies 256 bytes).
2. Changes can be made to the system dynamically without having to modify anything other than the Environment Descriptor.

The ZCPR3 utilities are much smaller and faster than their ZCPR2 ancestors. For a complete listing of all ZCPR3 utilities supplied with the distribution, see the appropriate section.

Software Required for Installation

Commercial Software. ZCPR3 is to be installed on a working CP/M 2.2 system. The commercial software required to do this installation is:

1. A working CP/M 2.2 System
2. Source to the BIOS of the target CP/M 2.2 System or an overlay patch for the Cold Boot Routine
3. The MAC assembler from Digital Research, Inc.
4. A debugger (DDT, SID, or equivalent) for performing the overlay procedures
5. A disk utility (SYSGEN or equivalent) for placing the operating system image onto the OS tracks on disk.

If the user desires to edit and reassemble the utilities, the Microsoft M80 assembler and L80 linker are also required.

System Segment Software. The following software is supplied with ZCPR3 and is required for installation:

Name of File	*Function*
ZCPR3.ASM	Source to the ZCPR3 Command Processor
Z3HDR.LIB	Configuration File read in by ZCPR3.ASM to tailor the ZCPR3 Command Processor
Z3BASE.LIB	Definition of the Memory Map of the ZCPR3 System to be created
SYSENV.ASM	ZCPR3 System Environment Descriptor Header for ZCPR3
SYSENV.LIB	System Environment Descriptor
SYSFCP.ASM	ZCPR3 Flow Command Package source
SYSFCP.LIB	Configuration File read in by SYSFCP.ASM to tailor the ZCPR3 Flow Command Package (this file may be derived from one of the SYSFCPn.LIB files below)
SYSIOP.ASM	ZCPR3 Input/Output Package source
SYSNDR.ASM	ZCPR3 Named Directory Definition File source Header for
SYSNDR.LIB	ZCPR3 Named Directory Definition
SYSRCP.ASM	ZCPR3 Resident Command Package source Header for ZCPR3
SYSRCP.LIB	Resident Command Package (this file may be derived from one of the SYSRCPn.LIB files below)

Other Useful Files

Name of File	*Function*
Z3LOC.COM	Utility to locate a CP/M CCP

Z3BASE1.LIB Z3BASE2.LIB	Sample ZCPR3 BASE files (Z3BASE.LIB)
Z3HDR1.LIB Z3HDR2.LIB	Sample ZCPR3 HDR files (Z3HDR.LIB)
SYSFCP1.LIB SYSFCP2.LIB	Sample ZCPR3 Flow Command Package headers
SYSRCP1.LIB SYSRCP2.LIB SYSRCP3.LIB SYSRCP4.LIB	Sample ZCPR3 Resident Command Package headers

Files Required for Installing ZEX. If the ZEX 3.0 Command File Processor is to be installed for use with the target ZCPR3 system, the following files are required. ZEX 3.1 and later is installed like the other utilities.

Name of File	*Function*
ZEX.ASM	Source of ZEX
ZEX.ZEX	ZEX Command File used to assemble new versions of ZEX once the first version is running
RELS.UTL	SID/ZSID Utility File (not supplied with ZCPR3)

Required Distribution Files. The following files are required for the installation of a complete ZCPR3 System.

Name of File	*Name of File*	*Name of File*
SYSENV.ASM	SYSFCP1.LIB	Z3BASE.LIB
SYSFCP.ASM	SYSFCP2.LIB	Z3BASE1.LIB
SYSIOP.ASM	SYSNDR.LIB	Z3BASE2.LIB
SYSNDR.ASM	SYSRCP1.LIB	Z3HDR.LIB
SYSRCP.ASM	SYSRCP2.LIB	Z3HDR1.LIB
ZCPR3.ASM	SYSRCP3.LIB	Z3HDR2.LIB
SYSENV.LIB	SYSRCP4.LIB	ZEX.ASM

Useful Distribution Files. The following files are useful, but not required, for the installation of a ZCPR3 System.

Name of File	*Name of File*	*Name of File*
Z3LOC.COM	Z3INS.COM	ZEX.ZEX

17 Selecting the Features

Features of ZCPR3

The installer must first decide what features the ZCPR3 System is to include, and his choices are:

1. Standard Overhead – Is the ZCPR3 System to include the standard 1K overhead or not? If not, which parts of the Standard Overhead are to be included?
2. Flow Command Package – Is the System to include Flow Commands or not?
3. Input/Output Package – Is the System to include I/O Packages or not?
4. Resident Command Package – Is the System to include Resident Commands or not?

Beyond these basic decisions, the contents of the following configuration files have to be determined:

1. Z3BASE.LIB – Base Addresses for the System
2. Z3HDR.LIB – Configuration Options for the ZCPR3 Command Processor
3. SYSFCP.LIB – Configuration Options for the Flow Command Packages (only if this feature is selected)
4. SYSRCP.LIB – Configuration Options for theResident Command Packages (only if this feature is selected)

Standard Overhead

The *Standard Overhead* of a ZCPR3 System consists of all buffers above 0F400H in Fig 16-2. These buffers contain:

1. External Stack
2. Command Line Buffer
3. Memory-Based Named Directory
4. External File Control Block
5. Message Buffers
6. Shell Stack
7. Environment Descriptor

The tradeoff analysis of whether to include these buffers or not follows. As a general recommendation, the features supported by these buffers are fundamental to the basic nature of ZCPR3 and it is recommended that all of the Standard Overhead be included. The cost of doing this is only 1K.

External Stack. The External Stack occupies 48 bytes, and its purpose is two-fold: 1) to free this amount of space within the ZCPR3 Command Processor for other purposes; and 2) to provide a common stack which can be easily accessed by the ZCPR3 utilities to restore system integrity when required. The external stack need not be initialized before use.

Command Line Buffer. The *Command Line Buffer* occupies just over 200 bytes, and is required by many functions of the ZCPR3 System. Its purpose is to store the command line entered from the console keyboard, passed by an executing SUBMIT file, or built by a ZCPR3 utility such as ALIAS or MENU. If this buffer is not supported externally (as recommended), then space for it will be taken up inside the ZCPR3 Command Processor. If made external to the ZCPR3 command processor, this buffer provides a mechanism to implement the following capabilities:

1. multiple commands on a single line, e.g., DIR;ERA *.BAK;DIR
2. certain useful front-ends, such as MENU
3. the ALIAS feature

The Command Line Buffer *must* be initialized before it is used. The first use is when the ZCPR3 Command Processor is first executed, so this initialization *must* be done during (or before, in some rare cases) the cold boot procedure in the BIOS.

Memory-Based Named Directory. The Memory-Based Named Directory contains the name-DU assignments for the named directories known to the system. Each name occupies 18 bytes, so the recommended allocation of 256 bytes can accommodate 14 names (18*14=252); this could be extended if more names are desired. This feature, while it finds immediate application on a hard disk system, is also convenient on a floppy-based system, and costs little.

The Memory-Based Named Directory buffer *must* be initialized before it is used. It is generally recommended that the Memory-Based Named Directory buffer be initialized during cold boot, but this is not mandatory. If the ZCPR3 command processor is set up to give precedence to the DU form over the DIR form, then this directory buffer may be initialized by a STARTUP alias (the command "STARTUP" is stored in the Command Line Buffer as a cold boot command). STARTUP may then run LDR, which will load an NDR (Named Directory) file.

External File Control Block. The *External File Control Block* occupies only 36 bytes (48 bytes were reserved for it in figure 16-3) and its purpose is two-fold: 1) to free space inside the ZCPR3 Command Processor; and 2) to provide a mechanism by which a utility can determine the name by which it was invoked. The ZCPR3 Command Processor stores the name of the command it just parsed into this buffer so that the command can read it and use it. Shells commonly use this feature to determine the name under which they were invoked so that they can set themselves up to be reexecuted.

No initialization is required for the External File Control Block.

Message Buffers. The *Message Buffers* of ZCPR3 occupy only 80 bytes, and they are very important as a mechanism through which the following operations can be performed: 1) ZCPR3 can leave messages about its own status, and these can be read by utilities invoked by ZCPR3; 2) programs can leave messages to give ZCPR3 instructions on how to perform certain operations, such as error handling and shell execution; and 3) one program can leave a message to be read and interpreted by another program that is executed later.

It cannot be emphasized enough that the ZCPR3 Message Buffers are *vital* to the operation of the system and should be always be included as a feature.

The Message Buffers *must* be initialized before they are used, and the ZCPR3 Command Processor begins using the Message Buffers immediately after cold boot.

Shell Stack. The *Shell Stack* permits the shell feature of ZCPR3 to be implemented, costs only 128 bytes, and also permits the shell feature to be extended to include invocation of one shell on top of another shell. For example, the MENU, SH, and VFILER utilities are invoked as shells under ZCPR3. Having a shell stack allows one shell, such as MENU, to invoke another shell, such as VFILER, causing MENU to be suspended. VFILER can run as long as desired; upon exit from VFILER, operation of MENU is resumed. Refer to Chapters 2 and 6 for user-oriented explanations of shells, and to Chapter 13 for their internal workings.

The Shell Stack must be initialized by the cold boot routine. Reasons are the same as for the Command Line Buffer.

Environment Descriptor. The *Environment Descriptor* (256 bytes) contains much detail on the ZCPR3 environment, including information on the features available and the TCAP entry (if any) for the user's CRT terminal. If the Environment Descriptor is not supported externally (as recommended), then each ZCPR3 utility must be assembled to include a copy of the ZCPR3 Environment Descriptor within it. If the Environment Descriptor is supported externally, each ZCPR3 utility contains only a pointer (2 bytes) to the descriptor. Installation of a utility then amounts merely to setting this pointer.

The ZCPR3 Environment Descriptor may be initialized either by the cold boot routine in the BIOS or by the execution of LDR on an ENV file as a STARTUP command.

Flow Command Packages. The *Flow Command Package* of ZCPR3 implements the basic flow constructs of the ZCPR3 System. These are the IF, ELSE, FI, and XIF commands. With this feature installed, command sequences like the following are possible:

```
IF EXIST MYFILE.TXT
      TYPE MYFILE.TXT
ELSE
      IF ERROR
            ECHO MYFILE.TXT NOT FOUND
      FI
FI
```

The Flow Command Package *must* be initialized by the cold boot routine. Reasons are the same as those for the Command Line Buffer.

Input/Output Packages. The *Input/Output Package* of ZCPR3 implements a set of Input/Output drivers which can be loaded dynamically to configure and extend the input/output system of the user's computer.

The Input/Output Package *must* be initialized by the cold boot routine. Reasons are the same as those for the Command Line Buffer.

Resident Command Packages. The *Resident Command Package* of ZCPR3 implements a set of commands which remain in memory until the RCP buffer is explicitly reloaded by the LDR.COM utility. These command packages extend the command set resident within the ZCPR3 Command Processor, and allow the user to change the command set from time to time.

The Resident Command Package *must* be initialized by the cold boot routine. Reasons are the same as those for the Command Line Buffer.

Other Buffers

External Path. The *External Path* (consisting of a few byte-pairs) is a buffer that contains the symbolic expression of the Command Search Path to be followed by the ZCPR3 command processor when searching for a COM file.

The External Path *must* be initialized by the cold boot routine.

Wheel Byte. The *Wheel Byte* (1 byte) is a flag read by some ZCPR3 utilities which defines the user to be privileged or not. If this byte is non-zero, the user is declared to be privileged, and certain functions are enabled which are not normally available to him. PWD (Print Working Directories), for example, will also display passwords to these directories if the user is priveleged and requests them.

The Wheel Byte should be initialized before invoking any utility that reads it.

18 Step 2 : Planning the ZCPR3 Memory Structure

The file Z3BASE.LIB defines the memory structure of the ZCPR3 System and serves as a common source of information about the target ZCPR3 System for all utilities and ZCPR3 System Segments. A number of the ZCPR3 system segments have "include" statements in their source, referencing this file, so that when they are assembled, adequate information on the memory structure of the system for which they are being assembled will be available to the assembler program.

Z3BASE.LIB is divided into two parts: 1) the comment header, which outlines the memory structure of the system in a manner similiar to Fig 16-1; and 2) the body, which contains a series of equates that define addresses of elements in the system, together with other information about various attributes of the system.

Z3BASE Header

Figure 18-1 shows the comment header of the example ZCPR3 System Z3BASE.LIB file. It is recommended that the installer fill out the details of the address range and features supported in the target ZCPR3 System in a Z3BASE.LIB file before doing any programming, and that he make a copy of Z3BASE.LIB as a reference for himself during the installation process.

```
;************************************************************
;*  Z3BASE.LIB -- Base Addresses for ZCPR3 System by R Conn *
;*                                                          *
;*  Address Range     Size  Function                        *
;*      0 -   FF     256 b  Standard CP/M Buffers except    *
;*     40 -   4A      11 b    for ZCPR3 External Path       *
;*     4B              1 b  Wheel Byte                      *
;*    100 - BFFF     ~48  K TPA                             *
;*   C000 - C7FF       2  K ZCPR3 Command Processor         *
;*   C800 - D5FF      3.5K  BDOSZ                           *
;*   D600 - E3FF      3.5K  CBIOSZ with Buffers             *
;*   E400 - EBFF       2  K Resident Command Package        *
;*   EC00 - F1FF      1.5K  Redirectable I/O Driver Package *
;*   F200 - F3FF      0.5K  Flow Command Package            *
;*   F400 - F4FF    256 b   Environment Descriptors         *
;*                          Bytes 00H-7FH:  Z3 Parameters   *
;*                          Bytes 80H-FFH:  Z3 Terminal Cap *
;*   F500 - F57F    128 b   ZCPR3 Shell Stack               *
;*   F580 - F5CF     80 b   ZCPR3 Message Buffers           *
;*                          Byte 0:  Error Flag (Z/NZ)      *
;*                          Byte 1:  IF (8 Levels)          *
;*                          Byte 2:  IF Active (8 Levels)   *
;*                          Byte 3:  Z3 Cmd Status          *
;*                                  00B - Normal            *
;*                              01B - Shell                   *
;*                              10B - Error                   *
;*                      Bytes 4&5: Error Address if 10B *
;*                      Byte 6: Program Error Code        *
;*                      Byte 7: ZEX Message Byte          *
;*                              00B - Normal              *
;*                              01B - Z3 Prompt           *
;*                              10B - Suspend Intercept *
;*                      Byte 8: ZEX Running Flag (0=No) *
;*                      Bytes 9-10: Address of Next       *
;*                             Char for ZEX to Return    *
;*                      Bytes 11-12: Address of First     *
;*                             Char in ZEX Memory-        *
;*                             Based File Buffer          *
;*                      Byte 13: SH Control Byte          *
;*                              Bit 0: Enable SHCMT       *
;*                              Bit 1: Enable SHECHO      *
;*                              Bit 7: Enable Shell       *
;*                                      Entry Wait        *
;*                      Bytes 14-15: Shell Scratch        *
;*                      Bytes 10H-2FH: Error Cmd          *
;*                      Bytes 30H-39H: Registers          *
;*                      Bytes 3AH-3FH: Reserved           *
;*                      Bytes 40H-4FH: User-Defined       *
;*   F5D0 - F5FF    48 b   ZCPR3 External FCB              *
;*   F600 - F6FF   256 b   Memory-Based Named Directory    *
;*   F700 - F7CF   208 b   Multiple Command Line Buffer    *
;*   F7D0 - F7FF    48 b   ZCPR3 External Stack            *
;*   F800 - FFFF     2 K   ROM                             *
;************************************************************
```

Figure 18-1. Z3BASE.LIB Comment Header

Z3BASE Body

The following paragraphs contain a reformatted duplicate of the body of the Z3BASE.LIB file. They provide additional information on how to set the equates. The installer may find it useful to keep the book open to these pages while editing the Z3BASE.LIB file.

Version Numbers, Memory Size, and Base. The following three equates define the version numbers of the ZCPR3 Command Processor and the CBIOSZ. Note that MSIZE explicitly states the size of the TPA, *not* total system size--the more usual meaning of this label (as, for instance, in Tarbell BIOS listings and those of some other manufacturers).

```
Z3REV   EQU     30        ; ZCPR3 REV NUMBER
CBREV   EQU     41        ; CBIOSZ REV NUMBER
MSIZE   EQU     48        ; SIZE OF TPA
```

These equates are usually referenced by the assembler to fill in these details in the Cold Boot signon message. They are not used by any ZCPR3 System Segments other than the BIOS.

The BASE equate specifies the base address of the user's CP/M system (normally 0 for the standard DRI version). This equate allows easy modification for non-standard versions of CP/M (e.g., H89 or Apple).

```
BASE    EQU     0
```

Processor Selection. The following equate selects the use of the 8080/8085 CPU or the Z80 CPU for the target ZCPR3. If your processor is an 8080 or 8085, set this equate to TRUE. If the processor is a Z80, set it to FALSE, since the Z80 code is much smaller and, by using relative jumps, more features can be packed into the available space.

```
I8080   EQU     FALSE
```

External Path. The following equates define the address of the ZCPR3 External Path and the number of two-byte elements contained in this path (maximum). If there is no ZCPR3 External Path, both of these values should be set to 0. ZCPR3 will then reserve space within itself for the command-search path.

```
EXPATH  EQU     40H       ; EXTERNAL PATH
EXPATHS EQU     5         ; 5 2-byte Path Elements
                          ; (PATH SIZE = EXPATHS*2 + 1)
```

Wheel Byte. The following equate defines the address of the ZCPR3 Wheel Byte.

```
Z3WHL   EQU     4BH       ; WHEEL BYTE ADDRESS
```

If there is no Wheel Byte, set this equate to 0. The C3H instruction (JMP) at memory location 0 will then be used as the Wheel Byte and, since this location is non-zero, the Wheel Byte will always be TRUE. If this equate is set to 0, do *not* provide the user with the commands to change this byte, since, by so doing, he will wipe out the warm boot jump at location 0.

CCP Location. The following equate defines the address of the ZCPR3 Command Processor. This address *must* be supplied.

```
CCP     EQU     0C000H    ; ZCPR3 COMMAND PROCESSOR
```

This value can be obtained by calculation or by using the Z3LOC utility.

RCP Location. The following equates define the address of the ZCPR3 Resident Command Package and its size in 128-byte blocks. If there is no ZCPR3 Resident Command Package, both of these values should be 0.

```
RCP      EQU    0E400H   ; RESIDENT COMMAND PACKAGE
RCPS     EQU    16       ; 16 128-byte Blocks (2K bytes)
```

IOP Location. The following equates define the address of the ZCPR3 Input/Output Package and its size in 128-byte blocks. If there is no ZCPR3 Input/Output Package, both of these values should be 0.

```
IOP      EQU    0EC00H   ; REDIRECTABLE I/O PACKAGE
IOPS     EQU    12       ; 12 128-byte Blocks (1.5K bytes)
```

FCP Location. The following equates define the address of the ZCPR3 Flow Command Package and its size in 128-byte blocks. If there is no ZCPR3 Flow Command Package, both of these values should be 0.

```
FCP      EQU    0F200H   ; FLOW COMMAND PACKAGE
FCPS     EQU    4        ; 4 128-byte Blocks (0.5K bytes)
```

ENV Location. The following equates define the address of the ZCPR3 Environment Descriptor and its size in 128-byte blocks. If there is no ZCPR3 Environment Descriptor, both of these values should be 0.

```
Z3ENV    EQU    0F400H   ; ENVIRONMENT DESCRIPTORS
Z3ENVS   EQU    2        ; SIZE OF DESCRIPTOR IN 128-BYTE BLOCKS
```

Shell Stack. The following equates define the address of the ZCPR3 Shell Stack, the number of entries permitted in the ZCPR3 Shell Stack, and the size of each entry in the Shell Stack in terms of bytes. If there is no ZCPR3 Shell Stack, all three values should be 0.

```
SHSTK    EQU    0F500H   ; ZCPR3 SHELL STACK
SHSTKS   EQU    4        ; NUMBER OF SHSIZE-BYTE SHELL STACK ENTRIES
SHSIZE   EQU    32       ; SIZE OF A SHELL STACK ENTRY
                         ;    (STACK SIZE = SHSTKS * SHSIZE)
```

The total amount of space occupied by the shell stack is SHSTKS*SHSIZE. In this configuration, 128 bytes are used (4*32).

ZCPR3 Messages. The following equate defines the address of the ZCPR3 Message Buffer. This buffer is always 80 bytes long. If there is no ZCPR3 Message Buffer, this address should be 0.

```
Z3MSG    EQU    0F580H   ; ZCPR3 MESSAGE BUFFER
```

External FCB. The following equate defines the address of the ZCPR3 External FCB. This buffer is always 36 bytes long. If there is no ZCPR3 External FCB, this address should be 0.

```
EXTFCB   EQU    0F5D0H   ; ZCPR3 EXTERNAL FCB
```

Named Directory Buffer. The following equates define the address and size (in terms of 18-byte entries) of the ZCPR3 Named Directory Buffer. If there is no such buffer, both of these values should be 0.

```
Z3NDIR   EQU     0F600H   ; ZCPR3 NAMED DIRECTORY AREA
Z3NDIRS  EQU     14       ; 14 18-byte Named Directory
                          ; Elements permitted.
                          ;(NDIR SIZE=Z3NDIRS*18+1 [for trailing 0])
```

Command Line Buffer. The following equates define the address and size (in bytes) of the ZCPR3 Command Line Buffer (formerly called the Multiple Command Line Buffer under ZCPR2). If there is no such buffer, both of these values should be 0.

```
Z3CL    EQU      0F700H   ; ZCPR3 COMMAND LINE BUFFER
Z3CLS   EQU      200      ; SIZE OF COMMAND LINE BUFFER
```

External Stack. The following equate defines the address of the ZCPR3 External Stack. This stack is always 48 bytes in size. If there is no such stack, this value should be 0.

```
EXTSTK EQU       0F7D0H   ; ZCPR3 EXTERNAL STACK
```

User Equates. The following equates are available for the implementer's target system. These are implementation-defined.

```
DJEPROM          EQU      0F800H  ; EPROM BASE ADDRESS
```

This is provided mainly as a convenience to the user. This value is used by my BOOT and BIOS, which also read this file for information.

19 Installation Steps 3—6

Step 3: Modifying the BIOS Cold Boot Routine

The following is a reformatted and edited copy of my BIOS (Basic Input/Output System) and the header file used to customize it. This information is provided as an example of how to modify the Cold Boot routine (labelled 'cboot' in this example) for a full installation of ZCPR3. Only the pertinent information is included.

CBIOSHDR.LIB—BIOS Configuration File

```
**************************************************************
*                                                            *
*  Control Processor/Microcomputer Basic I/O System          *
*   CP/ZM CBIOSZ Standard with CON:=CRT:                     *
*                                                            *
*   Customized for the ARIES-1 Microcomputer by              *
*           Richard Conn, 5 Jan 1984                         *
*                                                            *
**************************************************************

**************************************************************
*                                                            *
* The following revision number is in reference to the DR    *
* 2.8 CBIOS&.                                                *
*                                                            *
**************************************************************

revnum     equ     cbrev            ;CBIOSZ revision number
cpmrev     equ     z3rev            ;ZCPR3 revision number
```

The values 'cbrev' was provided by the BIOS source, and 'z3rev' came from Z3HDR.LIB.

<< Detail Left Out >>

```
cbdisk     equ     0F0H       ;Initial Disk to Log In, 0=A, 1=B
                              ;User 15, Disk A

**************************************************************
*                                                            *
* CP/M system equates. If reconfiguration of CP/M system     *
* is being done, the changes can be made to the following    *
* equates.                                                   *
*                                                         *
***********************************************************

bdos        equ    ccp+800h    ;BDOS address
bios        equ    ccp+1600h   ;CBIOS address

wbot        equ    0           ;Warm boot jump address
iobyte      equ    3           ;IOBYTE location
cdisk       equ    4           ;Address of last logged disk
entry       equ    5           ;BDOS entry jump address
buff        equ    80h         ;Default buffer address
tpa         equ    100h        ;Transient memory

retries     equ    10          ;Max retries on disk I/O before er
```

<< Detail Left Out >>

```
************************************************************
*                                                          *
*   Under the new redirectable I/O driver system, the I/O  *
* byte can be anything you desire. I have set it up as:    *
*                                                          *
*           ------------------------------                 *
*   IOBYTE  | LST:   | PUN: | RDR: | CON:  |               *
*           ------------------------------                 *
*    bits -> 7 6 5    4       3      2 1 0                 *
*                                                          *
*   The initial IOBYTE is currently defined as:            *
*           Bits 0-2: CON                                  *
*           Bit 3: RDR                                     *
*           Bit 4: PUN                                     *
*           Bits 5-7: LST                                  *
*                                                          *
*           CON: = CRT: CRT                                *
*           RDR: = CLOCK: System Clock                     *
*           PUN: = CLOCK: System Clock                     *
*           LST: = TTY: Printer                            *
*                                                          *
************************************************************

intioby      equ     000$1$1$001B    ; Initial IOBYTE
```

This I/O Byte could also be initialized within the initialization sequence in the Input/Output Package. Whenever LDR.COM loads an I/O Package, it calls an initialization routine.

```
****************************************************************
*                                                              *
* If there is a command inserted here, it will be given on     *
* cold boot.                                                   *
*   For Example:                                               *
*                                                              *
*   coldbeg db      'MBASIC MYPROG'                            *
*   coldend db      0                                          *
*                                                              *
* will execute microsoft basic, and mbasic will execute the    *
* "MYPROG" basic program.                                      *
*                                                              *
****************************************************************

acmd        macro         ;Define as Macro for Code Insertion
coldbeg:
    db      'STARTUP'     ;Cold boot command goes here
coldend:
    db      0
    endm
```

By ZCPR3 convention, STARTUP is a program created with the ALIAS utility. An Alias is a program created by the ZCPR3 ALIAS utility which generates command lines and places them into the Command Line Buffer of the ZCPR3 System. These command lines can be quite involved, including IF/ELSE constructs, and parameters may be passed into them at strategic points by an elaborate parameter passing mechanism. See the documentation (HLP file) on the ALIAS command for more details. The STARTUP Alias typically does not extract information from the command line and simply generates a sequence of commands which initialize the ZCPR3 System by running LDR on a variety of System Segments and performing other such operations. The Environment Descriptor is so key to ZCPR3 operation that if the cold boot routine does not already initialize the Environment Descriptor, it is recommended that the startup command line be prefixed with

```
LDR SYS.ENV
```

in order to ensure that the Environment Descriptor is properly loaded before anything else, including an Alias, runs. A valid, safe startup command line could be:

```
LDR SYS.ENV;STARTUP
```

```
************************************************************
*                                                          *
* Path to be Set for ZCPR2 on Cold Boot                    *
*                                                          *
************************************************************

idisk1      equ    'A'-'@' ;1st: Disk A, Current User
iuser1      equ    '$'
idisk2      equ    'A'-'@' ;2nd: Disk A, User 15
iuser2      equ    15
idisk3      equ    0     ;No 3rd Entry
iuser3      equ    0
idisk4      equ    0     ;No 4th Entry
iuser4      equ    0
```

<< Detail on I/O Devices Left Out >>

CBIOSZ—Selections from a ZCPR3 BIOS

```
* SYSTEM SEGMENT: CBIOSZ
* SYSTEM: ARIES-1
* CUSTOMIZED BY: RICHARD CONN

*----  Customize Section ----*
* Customization Performed in CBIOSHDR.LIB
*----  End of Customize Section ----*

************************************************************
*                                                          *
*  Control Processor/Microcomputer Basic I/O System        *
*   CP/ZM BIOSZ Standard with CON:=CRT:                    *
*   CHBIOSZ Body                                           *
*                                                          *
*   Customized for the ARIES-1 Microcomputer with Hard Disk *
*           by Richard Conn, Feb 2, 1984                   *
*                                                          *
************************************************************

;
; Macro Libraries for Customization
;
```

```
   MACLIB  Z3BASE
   MACLIB  CBIOSHDR
```

<< Detail Left Out >>

```
**************************************************************
*                                                            *
* The following are internal Cbios equates. Most are misc.   *
* constants.                                                 *
*                                                            *
**************************************************************

acr  equ    0dh          ;A carriage return
alf  equ    0ah          ;A line feed
XON  equ    11h          ;X-ON
XOFF equ    13h          ;X-OFF

**************************************************************
*                                                            *
* The jump table below must remain in the same order, the    *
* routines may be changed, but the function executed must    *
* be the same.                                               *
*                                                            *
**************************************************************

    org       bios           ;CBIOS starting address

    jmp       cboot          ;Cold boot entry point
wboote:
    jmp       wboot          ;Warm boot entry point
;
    jmp       const          ;Console status routine
    jmp       fconin         ;Console input
cout:
    jmp       fconout        ;Console output
    jmp       list           ;List device output
    jmp       punch          ;Punch device output
    jmp       reader         ;Reader device input
;
    jmp       home           ;Home drive
    jmp       setdrv         ;Select disk
    jmp      settrk        ;Set track
    jmp      setsec        ;Set sector
    jmp      setdma        ;Set DMA address
    jmp      read          ;Read the disk
    jmp      write         ;Write the disk
;
    jmp      listst        ;List device status
;
    jmp      sectran       ;Sector translation
;
    jmp      newio         ;Redirect I/O Drivers
```

My JMP table is extended slightly. NEWIO is for a system not described here.

```
************************************************************
*                                                          *
* These are the console I/O routines with buffer flush.    *
*                                                          *
************************************************************

fconin:
    call    flush         ;Flush Buffer
    jmp     conin

fconout:
    push    b             ;Save char
    call    flush         ;Flush Buffer
    pop     b             ;Get char
    jmp     conout

************************************************************
*                                                          *
* Gocpm is the entry point from cold boots, and warm boots. *
* It initializes some of the locations in page 0, and sets *
* up the initial DMA address (80h).                        *
*                                                          *
************************************************************

gocpm:
    call    const         ;Check for Input Char
    ora     a             ;NZ means char there
    cnz     conin           ;Flush it if so
;
    lxi     h,buff          ;Set up initial DMA address
    call    setdma
;
    mvi     a,(jmp)         ;Initialize jump to warm boot
    sta     wbot
    sta     entry           ;Initialize jump to BDOS
;
    lxi     h,wboote        ;Address in warm boot jump
    shld    wbot+1
;
    lxi     h,bdos+6        ;Address in BDOS jump
    shld    entry+1
;
    xra     a               ;A = 0
    sta     bufsec          ;Disk Jockey buffer empty
    sta     bufwrtn         ;Set buffer not dirty flag
;
    lda     cdisk           ;Jump to CP/M with currently
                            ;selected disk in C
    mov     c,a
```

This GOCPM section is more or less standard. The following section of GOCPM deals with initializing the Command Line Buffer on Cold Boot (or Warm Boot).

```
;
;  This code loads an optional command line on COLD BOOT only
;
    lda     cwflg           ;Test for any loaded command
    ora     a
    jnz     ccp+3           ;Enter ZCPR3 without command if Warm Boot
    lxi     d,coldbeg       ;Beginning of initial command
cldcmnd:
;
    if      z3cl ne 0       ;Multiple Commands Allowed?
;
    lxi     h,z3cl+4        ;Multiple Command buffer
;
    else
;
```

```
    lxi     h,ccp+8      ;Command buffer
;
    endif
;
cldl:
    ldax    d            ;Get char
    mov     m,a          ;Put char
    inx     h            ;Pt to next
    inx     d
    ora     a            ;Done?
    jrnz    cldl
    jmp     ccp          ;Run with Command

cwflg:
    db      0            ;Cold/warm boot flag

*************************************************************
*                                                           *
* If there is a command inserted here, it will be given if  *
* the auto feature is enabled.                              *
*   For Example:                                            *
*                                                           *
*   coldbeg db      'MBASIC MYPROG'                         *
*   coldend db      0                                       *
*                                                           *
* will execute microsoft basic, and mbasic will execute the *
* "MYPROG" basic program.                                   *
*                                                           *
*************************************************************

     acmd    ;Perform Macro from CBIOSHDR.LIB

*************************************************************
*                                                           *
* Signon message output during cold boot.                   *
*                                                           *
*************************************************************

prompt:
    db      acr,alf
    db      '0'+msize/10        ;CP/ZM memory size
    db      '0'+(msize mod 10)
    db      'K TPA ZCPR V'      ;ZCPR version number
    db      z3rev/10+'0','.',(z3rev mod 10)+'0'
    db      ', CBIOSZ V'        ;CBIOSZ version number
    db      cbrev/10+'0','.',(cbrev mod 10)+'0'

    if      first   ;if hard disk is A
    db      'H'  ;say this is a hard disk version
    else
    db      'F'  ;say this is a floppy version
    endif

    db      acr,alf
    db      0

************************************************************
*                                                          *
* Path for ZCPR 3.x initialized during cold boot.          *
*                                                          *
************************************************************
path:
    db      idisk1,iuser1   ;First Disk and User
    db      idisk2,iuser2   ;2nd Disk and User
    db      idisk3,iuser3   ;3rd Disk and User
    db      idisk4,iuser4   ;4th Disk and User

    db      0               ;End of PATH
```

This path is defined in CBIOSHDR.LIB.

```
************************************************************
*                                                          *
* Utility routine to output the message pointed at by  H&L,*
* terminated with a null.                                  *
*                                                          *
************************************************************

message:
    mov     a,m             ;Get a character of the message
    inx     h               ;Bump text pointer
```

```
   ana     a             ;Test for end
   rz                    ;Return if done
   push    h             ;Save pointer to text
   mov     c,a           ;Output character in C
   call    cout          ;Output the character
   pop     h             ;Restore the pointer
   jr      message       ;Continue until null reached

************************************************************
*                                                          *
* Cboot is the cold boot loader. All of CP/M has been loaded*
* when control is passed here.                             *
*                                                          *
************************************************************

cboot:
   lxi     sp,tpa          ;Set up stack
```

The following code segment copies the default command-search path into the External Path buffer. Since the ZCPR3 Command Processor uses this buffer the first time it searches for a COM file, it is recommended that this initialization always be done in the Cold Boot routine of the BIOS. There are some isolated cases where this is not required, but they will not be discussed here.

```
        if      expath ne 0     ;External Paths Supported
        lxi     d,path          ;Copy Cold-Boot Path
        lxi     h,expath        ;Into System External Path Area
        mvi     b,9             ;Always 9 bytes
        call    movlop
        endif
```

The following code segment initializes the Wheel Byte to non-priveleged status. This initialization may be done by a program executed by STARTUP if desired.

```
        if      z3whl ne 0      ;Wheel Byte Supported
        xra     a               ;Clear Wheel Byte
        sta     z3whl
        endif
```

The following code segment initializes the Resident Command Package buffer to zero. 128 bytes is larger than needed, but rather than specify the exact size for each package, this and the following initializations save code space in the BIOS and don't waste a significant amount of time. Since the ZCPR3 Command Processor uses the RCP before any COM files are executed, initialization of the RCP buffer is required to be performed in the BIOS Cold Boot routine.

```
	if	rcp ne 0	;RCPs Supported
	lxi	h,rcp		;RCP Address (zero fill)
	call	zero128		;128 bytes
	endif
```

The following code segment initializes the I/O Package with drivers which are contained within the BIOS. Later, when STARTUP executes, LDR will probably load a proper I/O Package over these drivers. Since I/O is used immediately after Cold Boot, this initialization must be performed in the Cold Boot routine.

```
	if	iop ne 0	;IOPs Supported
	lxi	d,iodrivers	;Set up I/O Drivers
	lxi	h,iop		;Location for drivers
	call	mover		;Copy an arbitrary 128 bytes
	else
	call	lstinit		;Init Simple LST Device
	call	clkinit		;Init Clock
	endif
```

The following code segment initializes the Flow Command Package. Like the RCP, the FCP must be initialized during Cold Boot.

```
	if	fcp ne 0	;FCPs Supported
	lxi	h,fcp		;FCP Address (zero fill)
	call	zero128		;128 bytes
	endif
```

The following code segment initializes the ZCPR3 Environment Descriptor. Since the first utility executed is usually STARTUP (which is an Alias), the Environment Descriptor must be initialized during Cold Boot.

```
	if	z3env ne 0	;ENVs Supported
	lxi	h,z3env		;ENV Address (zero fill)
	mvi	b,128+16	;128 bytes of environ + 16 bytes
	call	zerom		;of TCAP
	endif
```

The following code segment clears the Shell Stack. The ZCPR3 Command Processor queries this buffer almost immediately after Cold Boot, so this initialization must be performed during Cold Boot.

```
	if	shstk ne 0	;Shell Stack Supported
	xra	a		;Clear Stack
	sta	shstk
```

```
        endif
```

The following initialization must also be performed during Cold Boot since the ZCPR3 Command Processor uses messages extensively if this feature is enabled.

```
        if      z3msg ne 0      ;ZCPR3 Messages Supported
        lxi     h,z3msg         ;Clear Message Bytes
        mvi     b,80            ;80 bytes
        call    zerom
        endif
```

The following initialization is not required during Cold Boot if the analysis of the DU form is performed before the DIR form by the ZCPR3 Command Processor, but the installer is taking a risk that all will be well until LDR loads the Named Directory buffer if he does not perform the following initialization during Cold Boot.

```
        if      z3ndir ne 0     ;Named Directory Based in Memory
        lxi     h,z3ndir        ;Named Directory Base
        call    zero128         ;128 bytes
        endif
```

The following initialization of the Command Line buffer is absolutely required during Cold Boot. Only if the Command Line buffer is NOT external is this initialization unnecessary, but then a significant amount of the power of the ZCPR3 System is lost by having an internal Command Line buffer.

```
        if      z3cl ne 0       ;Multiple Commands Allowed
        lxi     d,cmdset        ;Set buffers for Multiple Comman
        lxi     h,z3cl          ;Command Line Base
        call    mover           ;Copy an arbitrary 128 bytes
        endif
```

As mentioned previously, the following initialization is not required and may be performed by the I/O Package loaded by LDR.

```
        if      iop ne 0
        mvi     a,intioby       ;Initialize the I/O Byte
        sta     iobyte
        endif
```

The following completes the Cold Boot initializations.

```
        lxi     h,prompt        ;Prep for sending signon message
        call    message         ;Send the prompt
        mvi     a,cbdisk        ;Select basic disk
        sta     cpmdrv
        sta     cdisk
```

<< Detail Left Out >>

```
    jmp     gocpm

*************************************************************
*                                                           *
* Mover moves 128 bytes of data. Source pointer in DE, Dest *
* pointer in HL.                                            *
*                                                           *
*************************************************************

mover:
    mvi     b,128           ;Length of transfer
movlop:
    ldax    d               ;Get a byte of source
    mov     m,a             ;Move it
    inx     d               ;Bump pointers
    inx     h
    djnz    movlop          ;Continue moving until done
    ret

zerofl      set        (rcp ne 0)or(fcp ne 0)or(z3env ne 0)
                            or(z3msg ne 0)
zerofl      set        zerofl or (z3ndir ne 0)
    if      zerofl
;
; Zero 128 bytes of memory pted to by HL
;
zero128:
    mvi     b,128           ;128 bytes
;
; Zero memory for B bytes   ; memory pted to by HL
;
zerom:
    mvi     m,0             ;store zero
    inx     h
    djnz    zerom
    ret
    endif
```

```
;
; Selection of LST: Device
;   If IOP is not supported, provide for one simple LST device;
; else, set LST device equal to console for later redefinition by
; loading an I/O Package via LDR
;
    if      iop ne 0
;
lstout      equ     djcout        ;same as console for now
punout      equ     djcout        ;same as console for now
rdrin       equ     djcin         ;same as console for now
;
    else
;
;  Initialize MPU Serial I/O Channel Characteristics and Baud Rat
;
lstinit:
```

<< Detail Left Out >>

```
;
    endif           ;IOP ne 0
;

*************************************************************
*                                                           *
* Primitive I/O Drivers which are loaded at Cold Boot time. *
*                                                           *
*************************************************************
uart        equ     origin+3F9H  ;UART address
rda         equ     4            ;UART RDA Bit
iodrivers:
;
    if      iop ne 0
;
    jr      ioerror       ;no Status Routine
    db      0             ;Fill 3 bytes
    jr      ioerror       ;no Select Routine
    db      0             ;Fill 3 bytes
    jr      ioerror       ;no Namer Routine
    db      0             ;Fill 3 bytes
```

```
;
    endif           ;iop ne 0
;
    ret                     ;Initialize Terminal
    db      0,0             ;Fill 3 bytes
    jr      ustat           ;Console Input Status
    db      0               ;Fill 3 bytes
    jmp     djcin           ;Console Input Char
    jmp     djcout          ;Console Output Char

    jmp     lstout          ;List Output Char

    jmp     punout          ;Punch Output Char

    jmp     rdrin           ;Reader Input Char
```

Note:
The above routines are located somewhere within the BIOS. They are very small and simple in nature, and they do not implement the I/O Byte. This is left for the redirectable I/O package which will be loaded later.

```
    mvi     a,0ffh          ;List Status Ready
    ora     a               ;Set Flags

    ret                     ;New I/O Driver Installation Routine

ioerror:
    xra     a               ;No device assignments
    ret

************************************************************
*                                                          *
* The following equates define the various Redirectable    *
* I/O routines in the SYSTEM I/O Area.                     *
*                                                          *
************************************************************

;
    if      iop ne 0
riobase     equ     iop+9           ;Relative I/O Base (less support)
```

```
    else
riobase     equ     iodrivers     ;Simple I/O Drivers
    endif           ;iop ne 0
;
tinit       equ     riobase       ;Terminal Init
const       equ     riobase+3     ;Console Input Status
conin       equ     riobase+6     ;Console Input
conout      equ     riobase+9     ;Console Output
list        equ     riobase+12    ;List Output
punch       equ     riobase+15    ;Punch Output
reader      equ     riobase+18    ;Reader Input
listst      equ     riobase+21    ;List Output Status
newio       equ     riobase+24    ;Redirectable I/O Patcher

*************************************************************
*                                                           *
* Initial Values for the External Command Line Buffers      *
* and Named Directory Memory-Based Buffers                  *
*                                                           *
*************************************************************

   if      z3cl ne 0
cmdset:
   dw      z3cl+4          ;Beginning of I/O Buffer
   db      z3cls           ;Size of I/O Buffer
   db      0               ;Empty Buffer
   db      0               ;Empty Buffer
   endif

*************************************************************
*                                                           *
* Wboot loads in all of CP/M except the CBIOS, then         *
* initializes system parameters as in cold boot. See the    *
* Cold Boot Loader listing for exactly what happens during  *
* warm and cold boots.                                      *
*                                                           *
*************************************************************

wboot:
```

<< Detail Left Out >>

Step 4: Editing Z3HDR.LIB

The following is a reformatted duplicate of the body of the Z3HDR.LIB file. It is provided here to present additional information on how to set the equates. It may be useful to the installer to have this installation manual open to these pages while he is editing the Z3HDR.LIB file.

The following is the Banner for Z3HDR.LIB:

```
Z3HDR - Maximum Configuration
Offset: 5100H
```

This offset is a note to the installer. It indicates the value to add to the R command of DDT in order to properly read in the HEX file of the ZCPR3 Command Processor into the Operating System memory image.

```
Module: Z3HDR
Author: Richard Conn
Module Used By: ZCPR3 Version 3.x
```

Note:

Z3HDR contains the key customization equates for ZCPR3. These equates allow the user to select various ZCPR3 options and do an extensive amount of tailoring of ZCPR3 to the user's desires.

Basic System Definitions

The following equates may be used to customize this CPR for the user's system and integration technique.

```
REL - TRUE if integration is to be done via MOVCPM
    - FALSE if integration is to be done via DDT and SYSGEN

CPRLOC - Base Page Address of CPR; this value can be obtained
      by running the CCPLOC program on your system, and if REL
      is FALSE, this value is supplied through the Z3BASE.LIB
         CCP equate

REL          EQU      FALSE
    IF       REL
CPRLOC       EQU      0
    ELSE
CPRLOC       EQU      CCP   ;VALUE PROVIDED IN Z3BASE.LIB
    ENDIF
```

CCPLOC.COM was an earlier version of Z3LOC.COM, which is now supplied with the ZCPR3 distribution. You may use either program to determine the base page of the CPR (Command Processor Replacement).

Integration by MOVCPM is not addressed here.

Default File Types

The following macros define the file types of the command object files (COM files under CP/M 2.2) to be loaded when a non-resident ZCPR3 command is given and of the indirect command files (SUB files under CP/M 2.2) to be used to extract commands from when the indirect command facility is invoked.

```
COMTYP      MACRO
            DB        'COM'
            ENDM

SUBTYP      MACRO
            DB        'SUB'
            ENDM
```

These equates are provided to allow the installer to select any file type he wishes for object and indirect command files. The indicated values of 'COM' and 'SUB' are conventional.

SUBMIT File Processing

The following flag enables the ability of ZCPR3 to process SUBMIT files (command files of the form $$$.SUB). If SUBON is TRUE, then ZCPR3 will process such files like CP/M's CCP normally does; if SUBON is FALSE, ZCPR3 will not process such files (ignore them). In such a case, only indirect command file facilities like ZEX will work. Much code is saved inside of the ZCPR3 Command Processor if SUBON is set to FALSE, but this rather useful facility is lost.

```
SUBON       EQU       TRUE
```

Command Prefix

The following flag allows ZCPR3 to accept commands of the form "du:command params" or "dir:command params". If DRVPREFIX is TRUE, this form is accepted; if FALSE, this form is not accepted.

```
DRVPREFIX   equ       TRUE
```

This option also affects arguments to commands, such as "DIR A5:*.TXT", and DU or DIR prefixes on these arguments are not processed either if DRVPREVIX is FALSE.

Command Attributes

The following equate allows the user to select the attributes of the COM files which are selected for execution. The ZCPR3 Command Processor can be made to execute only COM files with the System attribute set, with the Directory (non-System) attribute set, or with either attribute set. The following values are defined for this equate:

COMATT	Files Selected
0	System
80H	Directory
1	Both System and Directory

```
COMATT      equ      01H
```

ZCPR3 Resident Command Activation

The following equates enable various ZCPR3-resident commands. The user may invoke these as desired, but should keep in mind the size of the resulting ZCPR3 and make sure it does not exceed the required limits.

```
DIRON       equ      FALSE    ;DIR COMMAND
LTON        equ      FALSE    ;LIST, TYPE COMMANDS
GOON        equ      TRUE     ;GO COMMAND
ERAON       equ      FALSE    ;ERA COMMAND
SAVEON      equ      TRUE     ;SAVE COMMAND
RENON       equ      FALSE    ;REN COMMAND
GETON       equ      TRUE     ;GET COMMAND
JUMPON      equ      FALSE    ;JUMP COMMAND
NOTEON      equ      FALSE    ;NOTE COMMAND
```

Most of these commands are available as options in the Resident Command Packages. If selected for incorporation in the Resident Command Packages instead of the ZCPR3 Command Processor, space is freed in the ZCPR3 CPR and the functionality of these commands can be extended easily (eg, ERA in an RCP can have an Inspect option).

The Wheel equate table enables the WHEEL facility of ZCPR3. With this facility, a WHEEL BYTE, which exists somewhere in memory, is examined before a set of installer-selected commands are executed. If this byte is not zero, then the command proceeds. If it is zero, then the command is not allowed to proceed and is exited with an error message.

The following set of equates make each of the indicated commands selectable to respond to the Wheel Byte or not. For instance, if WERA=TRUE, then it responds to the Wheel Byte; if WERA=FALSE, it does not.

```
   IF    Z3WHL NE 0       ;IF A WHEEL BYTE ADDRESS IS DEFINED
WERA     equ      FALSE   ;Make ERA a Wheel-Oriented Command
WREN     equ      FALSE   ; " REN   "   "     "     "
WLT      equ      FALSE   ; " L/T   "   "     "     " (LIST/TYPE)
WGO      equ      FALSE   ; " GO    "   "     "     "
WSAVE    equ      FALSE   ; " SAVE  "   "     "     "
WGET     equ      FALSE   ; " GET   "   "     "     "
WJUMP    equ      FALSE   ; " JUMP  "   "     "     "
WDU      equ      FALSE   ; " DU:   "   "     "     " (DU/DIR Change
WHEEL    set      WERA OR WREN OR WLT OR WGO OR WSAVE OR WGET
WHEEL    set      WHEEL OR WJUMP OR WDU
         ENDIF            ;Z3WHL
```

The commands inside of the Resident Command Package can also be set to respond to the Wheel Byte.

ZCPR3 Resident Command Table

This table consists of the names of the various ZCPR3-resident commands and their addresses. The NCHARS equate defines how many characters long each name may be, and all table entries must be exactly the indicated number of characters (trailing spaces are used to fill out shorter names).

Each table entry is structured as follows:

```
    DB          'CMND'   ;Name of Command (NCHARS long)
    DB          CMNDADR ;Address of Command within ZCPR3
```

The installer should only change the names of the commands as desired and should not, as a rule, touch the address definition since this is fixed within the body of ZCPR3.

```
NCHARS          EQU         4    ;NUMBER OF CHARS/COMMAND

CTABLE          MACRO
;
      IF        DIRON
      DB        'DIR '
      DW        DIR                 ;DIRECTORY DISPLAY COMMAND
      ENDIF
;
      IF        LTON
      DB        'LIST'
      DW        LIST                ;LIST FILE ON PRINTER COMMAND
      DB        'TYPE'
      DW        TYPE                ;TYPE FILE ON CONSOLE COMMAND
      ENDIF
;
      IF        GOON
      DB        'GO  '
      DW        GO                  ;EXECUTE CURRENT TPA COMMAND
      ENDIF
;
      IF        ERAON
      DB        'ERA '
      DW        ERA                 ;ERASE FILES COMMAND
      ENDIF
;
      IF        SAVEON
```

```
    DB      'SAVE'
    DW      SAVE            ;SAVE TPA COMMAND
    ENDIF
;
    IF      RENON
    DB      'REN '
    DW      REN             ;RENAME FILES COMMAND
    ENDIF
;
    IF      GETON
    DB      'GET '
    DW      GET             ;LOAD FILE INTO TPA COMMAND
    ENDIF
;
    IF      JUMPON
    DB      'JUMP'
    DW      JUMP            ;JUMP TO ANY MEMORY LOCATION COMMAND
    ENDIF
;
    IF      NOTEON
    DB      'NOTE'
    DW      NOTE            ;NOTE - NULL COMMAND (NOP)
    ENDIF
;
    ENDM
```

Controls on ZCPR3 Resident Commands

The following sets of equates provide special controls and parameters on various ZCPR3-resident commands.

The following equates set the width of the spacing between the file names for the DIR command and the character used to separate file names from one another on the same line.

Assuming that FENCE is set to the character '|', If WIDE is TRUE, then the output will look like:

    filename.typ___|___filename.typ ...

while if WIDE is FALSE, the output will look like:

    filename.typ_|_filename.typ ...

(underscore represents a space)

```
WIDE        EQU     TRUE
FENCE       EQU     '|'
```

The WIDE equate is intended to provide for shorter lines for those users with 64-column displays. For those with 80-column displays, the output lines are much easier to read.

The following equates define two flags which are used in conjunction with the DIR command on the command line. SYSFLG is the character used to indicate to DIR that all files, both System and non-System, are to be displayed. SOFLG is the character used to indicate to DIR that only the System files are to be displayed. By default, DIR displays non-System files.

For example, if SYSFLG is set to 'A' and SOFLG is set to 'S', then:

```
DIR *.COM A
```

displays all COM files with both System and non-System attributes while:

```
DIR *.COM S
```

displays only COM files with the System attribute. Naturally:

```
DIR *.COM
```

displays only COM files with the non-System attribute.

```
SYSFLG      EQU   'A'
SOFLG       EQU    'S'
```

The following equate causes ERA to confirm the files to be erased before it goes ahead and erases them. If ERAOK is TRUE, then the user will be prompted each time; if it is FALSE, then the user will not be prompted.

```
ERAOK       equ     FALSE
```

If ERAOK is TRUE, the following equate adds a Verify option to the ERA command which causes the user to be prompted only if the Verify option letter, defined by ERDFLG, is given after the file name. If ERAV is TRUE, then the user will be asked to verify only when ERDFLG is contained in the command line; if ERAV is FALSE, the user will always be asked to verify.

For example, if ERAOK is TRUE, ERAV is TRUE, and ERDFLG is 'V', then the command:

```
ERA *.* V
```

will result in the file names being displayed and the user being asked for verification. If the V option were not given, the user would not be asked for verification.

```
ERAV        equ     FALSE
ERDFLG      equ     'V'
```

The following equates set the paging parameters for the TYPE command.

PGDFLT determines if TYPE pages by default. If PGDFLT is TRUE, then:

```
    TYPE FILE.TXT
```

will be paged. If PGDFLT is FALSE, the above command will not be paged.

PGDFLG defines the option character in the TYPE command line which is used to toggle the default set by PGDFLT. Assuming that PGDFLG is set to 'P', then:

```
    TYPE FILE.TXT P
```

will page the file listing if PGDFLT is FALSE and not page it if PGDFLT is TRUE.

```
    PGDFLT        EQU       TRUE
    PGDFLG        EQU       'P'
```

The following equate defines the number of lines on the user's CRT screen for use by the TYPE command when it is paging. This value is usually 24.

```
    NLINES        EQU       24
```

The following equate defines the option letter used with the SAVE command to indicate that the associated number is 128-byte sectors as opposed to 256-byte pages. For example, if SECTFLG is set to 'S', then:

```
    SAVE 25 FILE.BIN S
```

save 25 128-byte sectors starting at location 100H into the file named FILE.BIN. If the S option was not present, SAVE would have saved 25 256-byte blocks starting at location 100H into the file named FILE.BIN.

```
    SECTFLG       EQU       'S'
```

Path Definition

The following equate specifies the address of the PATH to be followed for the PATH command-search if the PATH is to be initialized by the BIOS and set by the user via a PATH.COM program. The value of PATH should be the address of the PATH data area in memory. If the internal PATH provided by ZCPR3 is to be used, then PATHBASE should be equated to 0, which selects the PATH located just after the MEMLOAD routine. If the external PATH is to be used, then PATHBASE should be set to the address of the external path.

A PATH is a series of byte-pairs, terminated by a binary 0. The first byte of each pair is the disk number (1-16 for disks A-P), and the second byte of each pair is the user number (0-31). The special character '$' indicates the current user or current disk. For example, the path from current disk/current user to current disk/user 0 to disk A/user 0 is selected by the following sequence:

```
            DB      '$$'    ;current disk/user
            DB      '$',0   ;current disk/user 0
            DB      1,0     ;disk A/user 0
            DB      0       ;end of path
    IF      EXPATH NE 0     ;External Path Selected
```

This equate defines the base address of the external path

```
PATH        equ      EXPATH      ;External ZCPR3 PATH

            ELSE                 ;Internal Path Selected
```

The following macro defines the n-element internal path

```
IPATH       MACRO
    db      'A'-'@','$'     ;Disk A, Current User
    db      'A'-'@',0       ;Disk A, User 0
    db      0               ;End of Path -- MUST be here
    ENDM

    ENDIF
```

The following flag enables ZCPR3 to perform an optimized path search when it is searching along a path for a file. If this equate is TRUE, ZCPR3 will build a path in memory of absolute entries (A1, B7, etc) from the symbolic path (one containing '$') which is the path it would otherwise use. This new path would contain no duplicate path elements, where a symbolic path analysis may. For example, if the path is:

```
db      'A'-'@','$'     ;disk A, current user
db      'A'-'@',15      ;disk A, user 15
db      0
```

then if the user is logged into A15, setting the below equate to TRUE would allow ZCPR3 to build the path:

```
db      'A'-'@',15      ;only one entry
db      0
```

in the analysis of this symbolic path, while with this equate FALSE, ZCPR3 may log into A15 as many as three times (once for the default and twice more for the symbolic path) in looking for a file which is not found before it gives up. Using this minimum path facility costs some code in ZCPR3, but it speeds up processing noticeably in some cases.

Enable this equate if MINIMUM PATH SEARCH is to be employed.

```
MINPATH     EQU     TRUE
```

In searching for a file along a path, ZCPR3 can be commanded to always look in the current logged-in directory before beginning the path search. This equate controls this feature. If SCANCUR is set to TRUE, the current directory need never be referenced in a symbolic path expression (DB '$','$') since SCANCUR insures that the current directory is scanned.

Enable this equate if the current DU is always to be scanned.

```
SCANCUR         EQU        TRUE
```

DU and DIR Controls

The following equate enables the appearance of the current disk/user in the ZCPR3 prompt. If set to FALSE, the prompt appears as '>' (assuming > is the current value of CPRMPT). If set to TRUE, the prompt appears as 'd>' or 'dn>' (see INCLNDR below).

```
INCLDU          equ        TRUE
```

The following equate allows ZCPR3 to accept the DU: prefix or login form for input. Set this to TRUE if DU: prefix is to be allowed.

Setting this equate to TRUE allows the following forms:

```
A>B1:
A>TYPE B4:FILE.TXT
A>B:
A>1:
```

```
ACCPTDU         EQU        TRUE
```

This equate enables ZCPR3 to process DIR: forms internally through the memory-based named directory buffer. This equate and the NDBASE address should be TRUE (non-zero) in order to enable ZCPR3 to process named directories.

If NDINCP is TRUE, the following forms are allowed:

```
A>ROOT:
A>TYPE TEXT:FILE.TXT
```

if the other associated equates (below) are set correctly.

```
NDINCP          EQU        TRUE
```

The following equate will cause the name of the current directory to be displayed as part of the prompt along with the DU form if enabled (see INCLDU above).

For example, if INCLNDR is TRUE, the prompt would look like:

```
B7:TEXT>            -- if INCLDU is also TRUE
TEXT>               -- if INCLDU is FALSE
```

```
INCLNDR         EQU        TRUE
```

The following equate allows ZCPR3 to accept the DIR: prefix or login form for input. Set this to TRUE if DIR: prefix is to be allowed.

Setting this equate to TRUE allows the following forms:

```
                    A>ROOT:
                    A>TYPE TEXT:FILE.TXT

ACCPTND             EQU         TRUE
```

The following equate determines the hierarchy of DU:/DIR: evaluation. Set this to TRUE if DU: is to be tested for before DIR: or set this to FALSE if DIR: is to be tested for before DU:. If this is FALSE, named directories like C: (standing for C work area–NOT disk C) are permitted.

Assuming that a directory for C programs, named 'C', and a root directory, named 'ROOT', exist, then if DUFIRST is set to FALSE:

```
A>C:    -- logs the user into the directory named 'C'
A>ROOT: -- logs the user into the directory named 'ROOT'
```

while if DUFIRST is set to TRUE:

```
A>C:    -- logs the user into disk C:
                  (dir C can't be accessed)
A>ROOT: -- logs the user into the directory named 'ROOT'
```

```
DUFIRST             EQU          FALSE
```

Enable password check on named directory references. If a named directory is referenced and has a password associated with it, ZCPR3 will ask the user for this password and approve the reference only if he gives a valid response. One and only one try is permitted. Setting this equate to TRUE will enable the password check facility.

```
PWCHECK             EQU          TRUE
```

Command Line Buffer Control

The MULTCMD equate enables the feature of having more than one command on the same line, separated by a separation char which is defined by the CMDSEP equate. If this feature is enabled, the command line buffer and buffer pointers are moved outside of ZCPR3 at the indicated address of Z3CL.

MULTCMD indicates if the ability to have more than one command on a line is to be enabled, and CMDSEP is the character used to separate these commands. For example, if CMDSEP is ';' and MULTCMD is TRUE, then commands like this are possible:

```
                    ERA *.BAK;DIR

                    IF           Z3CL NE 0
MULTCMD             equ          TRUE
                    ELSE
MULTCMD             equ          FALSE
```

```
                ENDIF
CMDSEP          equ         ';'
```

CMDRUN—Extended Command Processing

This equate enables the ZCPR3 CMDRUN facility. If CMDRUN is TRUE, then another stage of command processing is invoked should ZCPR3 fail to find a COM file when the user gives a command. This stage involves invoking the COM file specified by CMDFCB and giving it the current command line as an argument. In this way, if, say, M80 PROG2 fails as a command, a new command like LRUNZ M80 PROG2, SUB M80 PROG2, or ZEX M80 PROG2 may be processed. If the new command fails, an appropriate error message is given.

The ROOTONLY option causes ZCPR3 to only look at the Root (bottom of path) for the Extended Command Processor if it is set to TRUE. If it is set to FALSE, the path is searched for the Extended Command Processor. The tradeoff here is that ROOTONLY = TRUE is less flexible but somewhat faster than ROOTONLY = FALSE.

```
CMDRUN      equ     FALSE     ; Enable the Facility
            if      CMDRUN
ROOTONLY    equu    TRUE      ; TRUE if look at Root Only for
                              ; Extended Command Processor,
                              ; FALSE if look along path
CMDFCB      MACRO
            db      0
            db      'CMDRUN'      ;Name of Program
            db      'COM'         ;File Type
            ENDM
            endif  ;CMDRUN
```

Flow Command Facility

This equate enables ZCPR3 to respond to IF processing. ZCPR3 simply flushes commands if a FALSE IF is currently engaged. FCPs must be enabled for IFON to work correctly.

```
IFON            EQU         TRUE
```

Miscellaneous Equates

```
MAXUSR      EQU     31    ;MAXIMUM USER NUMBER ACCESSIBLE
MAXDISK     EQU     4     ;MAXIMUM NUMBER OF DISKS ACCESSIBLE
```

The DU form will only be allowed to reference disks and user areas up to the indicated maximum values. These limits, however, do not apply to named directories. For instance, if MAXUSR was 20 and the name SPECIAL was equated to B31, then 'SPECIAL:' would be resolved correctly but 'B31:' would cause an error. Named directories can have password protection associated with them, so B31 can be a directory which is accessed only if the user knows the password for it.

```
SUPRES          EQU         TRUE    ;SUPPRESSES USER # REPORT FOR USER 0
```

If you are logged into B0:, then the prompt would look something like:

```
            B>          if SUPRES is TRUE
            B0>         if SUPRES is FALSE
```

```
SPRMPT          EQU         '$'     ;CPR PROMPT INDICATING SUBMIT COMMAND
CPRMPTEQU       '>'                 ;CPR PROMPT INDICATING USER COMMAND
```

With these values:

```
        B1>         appears for commands typed by the user
        B1$         appears for commands input from a submit file
```

```
NUMBASE         EQU         'H'     ;CHAR USED TO SWITCH FROM DEFAULT
                                    ;NUMBER BASE
```

This option applies to the SAVE command only. It permits forms like:

```
            SAVE 17 myfile.bin      17 is decimal
            SAVE 11H myfile.bin     11H is hexadecimal
```

This feature is handy in that values output by tools like DDT do not need to be converted to hexadecimal before the SAVE command can be used.

```
CURIND          EQU         '$'     ;SYMBOL FOR CURRENT DISK OR USER

COMMENT         EQU         ';'     ;LINES BEGINNING WITH THIS CHAR
                                    ;ARE COMMENTS
```

Step 5: Overlaying the old BIOS and the CCP

The procedure for overlaying the old BIOS is the same as with conventional CP/M. The relative offset for the ZCPR3 Command Processor Replacement will be the same as that used for the BIOS. Refer to the manual published by Digital Research for more detail.

An example of this procedure is provided in the sample session which follows.

Step 6: Implanting the Operating System Image

Once everything is overlayed in the memory image of the system, SYSGEN is usually used to implant the operating system image on the system tracks of the disk. Refer to the manual published by Digital Research for more detail.

An example of this procedure is provided in the sample session which follows.

Sample Session

The following directory display shows the files we will be working with. Note that Z3BASE.LIB and Z3HDR.LIB are instrumental to the installation of almost all of the System Segments—Z3BASE.LIB and Z3HDR.LIB are read in by the MAC assembler during the assembly process.

```
B11>xd /oa
XD III Version 1.2
Filename.Typ Size K  Filename.Typ Size K  Filename.Typ Size K
-------- --- ------  -------- --- ------  -------- --- ------
CBIOSZ   .ASM    56  SYSRCP   .ASM    44  SYSNDR   .LIB     4
SYSENV   .ASM     4  ZCPR3    .ASM    68  SYSRCP   .LIB    12
SYSFCP   .ASM    20  CBIOSHDR.LIB     12  Z3BASE   .LIB    12
SYSIOP   .ASM    32  SYSENV   .LIB     4  Z3HDR    .LIB    20
SYSNDR   .ASM     4  SYSFCP   .LIB     8
     B 11:  --   14  Files Using  300K ( 2328K Left)
```

Assembling SYSENV

The following illustrates the assembly of SYSENV to produce the SYS.ENV file. This file will be the ZCPR3 System Environment Descriptor.

```
B11>zex mac sysenv
ZEX, Version 3.0
B11> ZEX: ;
B11> ZEX: ;  MAC -- CP/M Standard MACRO Assembler and Loader
B11> ZEX: ;
B11> ZEX: ;     Suppress FALSE IF Printout
B11> ZEX: ;
B11> ZEX: IF NUL SYSENV
B11> ZEX: MAC SYSENV $-S PZ
CP/M MACRO ASSEM 2.0
0200
01AH USE FACTOR
END OF ASSEMBLY

B11> ZEX: IF INPUT Abort if Errors Exist
IF True?
B11> ZEX: ERA SYSENV.BAK  ;NOTE Cleanup
 No Files

B11> ZEX: ERA SYSENV.COM
 No Files
B11> ZEX: MLOAD SYSENV    ;NOTE Load Hex File
MLOAD ver. 1.4   Copyright (C) 1983 Ronald G. Fowler
Loaded 172 bytes (00ACH - 2 records) to file B:SYSENV.COM
Start address: 0100H Ending address: 01ADH Bias: 0000H
```

```
B11> ZEX: FI
B11> ZEX: ERA SYSENV.HEX
 SYSENV  .HEX
B11> ZEX: FI
B11> ZEX: ;
B11> ZEX: ;  Assembly Complete
B11> ZEX: ;
B11> ZEX: Done>
B11>ren sys.env=sysenv.com
```

Assembling SYSNDR

```
B11>zex mac sysndr
ZEX, Version 3.0
B11> ZEX: ;
B11> ZEX: ;  MAC -- CP/M Standard MACRO Assembler and Loader
B11> ZEX: ;
B11> ZEX: ;     Suppress FALSE IF Printout
B11> ZEX: ;
B11> ZEX: IF NUL SYSNDR
B11> ZEX: MAC SYSNDR $-S PZ
CP/M MACRO ASSEM 2.0
01EB
005H USE FACTOR
END OF ASSEMBLY

B11> ZEX: IF INPUT Abort if Errors Exist
IF True?
B11> ZEX: ERA SYSNDR.BAK  ;NOTE Cleanup
 No Files

B11> ZEX: ERA SYSNDR.COM
 No Files
B11> ZEX: MLOAD SYSNDR    ;NOTE Load Hex File
MLOAD ver. 1.4   Copyright (C) 1983 Ronald G. Fowler
Loaded 235 bytes (00EBH - 2 records) to file B:SYSNDR.COM
Start address: 0100H Ending address: 01EAH Bias: 0000H

B11> ZEX: FI
B11> ZEX: ERA SYSNDR.HEX
```

```
 SYSNDR  .HEX
B11> ZEX: FI
B11> ZEX: ;
B11> ZEX: ;  Assembly Complete
B11> ZEX: ;
B11> ZEX: Done>
B11>ren sys.ndr=sysndr.com
```

Assembling SYSIOP

```
B11>zex mac sysiop
ZEX, Version 3.0
B11> ZEX: ;
B11> ZEX: ; MAC -- CP/M Standard MACRO Assembler and Loader
B11> ZEX: ;
B11> ZEX: ; Suppress FALSE IF Printout
B11> ZEX: ;
B11> ZEX: IF NUL SYSIOP
B11> ZEX: MAC SYSIOP $-S PZ
CP/M MACRO ASSEM 2.0
F0AA
016H USE FACTOR
END OF ASSEMBLY

B11> ZEX: IF INPUT Abort if Errors Exist
IF True?
B11> ZEX: ERA SYSIOP.BAK ;NOTE Cleanup
 No Files

B11> ZEX: ERA SYSIOP.COM
 No Files
B11> ZEX: MLOAD SYSIOP ;NOTE Load Hex File
MLOAD ver. 1.4 Copyright (C) 1983 Ronald G. Fowler
Loaded 1192 bytes (04A8H - 10 records) to file B:SYSIOP.COM
Start address: EC00H Ending address: F0A7H Bias: 0000H

++ Warning: program origin NOT at 100H ++

B11> ZEX: FI
B11> ZEX: ERA SYSIOP.HEX
 SYSIOP .HEX
```

```
B11> ZEX: FI
B11> ZEX: ;
B11> ZEX: ; Assembly Complete
B11> ZEX: ;
B11> ZEX: Done>
B11>ren sys.iop=sysiop.com
```

Assembling SYSRCP

```
B11>zex mac sysrcp
ZEX, Version 3.0
B11> ZEX: ;
B11> ZEX: ;  MAC -- CP/M Standard MACRO Assembler and Loader
B11> ZEX: ;
B11> ZEX: ;     Suppress FALSE IF Printout
B11> ZEX: ;
B11> ZEX: IF NUL SYSRCP
B11> ZEX: MAC SYSRCP $-S PZ
CP/M MACRO ASSEM 2.0
EBF1
017H USE FACTOR
END OF ASSEMBLY

B11> ZEX: IF INPUT Abort if Errors Exist
IF True?
B11> ZEX: ERA SYSRCP.BAK  ;NOTE Cleanup
 No Files

B11> ZEX: ERA SYSRCP.COM
 No Files
B11> ZEX: MLOAD SYSRCP    ;NOTE Load Hex File
MLOAD ver. 1.4   Copyright (C) 1983 Ronald G. Fowler
Loaded 2027 bytes (07EBH - 16 records) to file B:SYSRCP.COM
Start address: E400H Ending address: EBEEH Bias: 0000H

++ Warning: program origin NOT at 100H ++

B11> ZEX: FI
B11> ZEX: ERA SYSRCP.HEX
 SYSRCP  .HEX
B11> ZEX: FI
```

```
B11> ZEX: ;
B11> ZEX: ;  Assembly Complete
B11> ZEX: ;
B11> ZEX: Done>
B11>ren sys.rcp=sysrcp.com
```

Assembling SYSFCP

```
B11>zex mac sysfcp
ZEX, Version 3.0
B11> ZEX: ;
B11> ZEX: ;  MAC -- CP/M Standard MACRO Assembler and Loader
B11> ZEX: ;
B11> ZEX: ;     Suppress FALSE IF Printout
B11> ZEX: ;
B11> ZEX: IF NUL SYSFCP
B11> ZEX: MAC SYSFCP $-S PZ
CP/M MACRO ASSEM 2.0
F3F3
00CH USE FACTOR
END OF ASSEMBLY

B11> ZEX: IF INPUT Abort if Errors Exist
IF True?
B11> ZEX: ERA SYSFCP.BAK   ;NOTE Cleanup
 No Files

B11> ZEX: ERA SYSFCP.COM
 No Files
B11> ZEX: MLOAD SYSFCP  ;NOTE Load Hex File
MLOAD ver. 1.4   Copyright (C) 1983 Ronald G. Fowler
Loaded 499 bytes (01F3H - 4 records) to file B:SYSFCP.COM
Start address: F200H Ending address: F3F2H Bias: 0000H

++ Warning: program origin NOT at 100H ++

B11> ZEX: FI
B11> ZEX: ERA SYSFCP.HEX
 SYSFCP  .HEX
B11> ZEX: FI
B11> ZEX: ;
```

```
B11> ZEX: ;  Assembly Complete
B11> ZEX: ;
B11> ZEX: Done>
B11>ren sys.fcp=sysfcp.com
```

Creating MYTERM.Z3T via TCSELECT

```
B11>tcselect myterm
TCSELECT, Version 1.0

** Terminal Menu 1 for Z3TCAP Version 1.1 **

A.  AA Ambassador             K.  Concept 100
B.  ADDS Consul 980           L.  Concept 108
C.  ADDS Regent 20            M.  CT82
D.  ADDS Viewpoint            N.  DEC VT52
E.  ADM 2                     O.  DEC VT100
F.  ADM 31                    P.  Dialogue 80
G.  ADM 3A                    Q.  Direct 800/A
H.  ADM 42                    R.  General Trm 100A
I.  Bantam 550                S.  Hazeltine 1420
J.  CDC 456                   T.  Hazeltine 1500

Enter Selection, + for Next, or ^C to Exit - +

** Terminal Menu 2 for Z3TCAP Version 1.1 **

A.  Hazeltine 1510            K. P Elmer 1200
B.  Hazeltine 1520            L. SOROC 120
C.  H19 (ANSI Mode)           M. Super Bee
D.  H19 (Heath Mode)          N. TAB 132
E.  HP 2621                   O. Teleray 1061
F.  IBM 3101                  P. Teleray 3800
G.  Micro Bee                 Q. TTY 4424
H.  Microterm ACT IV          R. TVI 912
I.  Microterm ACT V           S. TVI 920
J.  P Elmer 1100              T. TVI 950

Enter Selection, - for Last, + for Next, or ^C to Exit - T

  Selected Terminal is: TVI 950        -- Confirm (Y/N)? Y
```

```
File MYTERM  .Z3T Created
```

Recap

To recap, the main System Segments which will be loaded by the LDR utility have now been created. These System Segments are:

```
B11>xd sys.* oa
XD III Version 1.2
Filename.Typ Size K  Filename.Typ Size K  Filename.Typ Size K
-------- --- ------  -------- --- ------  -------- --- ------
SYS     .ENV      4  SYS     .IOP      4  SYS     .RCP      4
SYS     .FCP      4  SYS     .NDR      4
    B 11: --      5 Files Using    20K ( 2304K Left)

B11>xd myterm.z3t oa
XD III  Version 1.2
Filename.Typ Size K  Filename.Typ Size K  Filename.Typ Size K
-------- --- ------  -------- --- ------  -------- --- ------
MYTERM  .Z3T 4
    B 11: - 1 Files Using 4K ( 2304K Left)
```

Assembling the CBIOS

The following illustrates the assembly of the Customized Basic Input/Output System (CBIOS). CBIOS has already been customized so that its Cold Boot routine performs the proper initializations of buffers required by the ZCPR3 System.

```
    B11>mac cbiosz $-s pz
    CP/M MACRO ASSEM 2.0
    E3CE
    01FH USE FACTOR
    END OF ASSEMBLY
```

Assembling the ZCPR3 CPR

The following illustrates the assembly of the ZCPR3 Command Processor Replacement. All customization has been done in the files Z3BASE.LIB and Z3HDR.LIB.

```
    B11>mac zcpr3 $-s pz
    CP/M MACRO ASSEM 2.0
    C7E9
    01EH USE FACTOR
    END OF ASSEMBLY
```

Obtaining the Operating System Image

The following SYSGEN pulls the Operating System Image off of the System Tracks. If you are installing a ZCPR3 System from scratch, you should be running a CP/M system which has been moved down to allow space for the ZCPR3 buffers in high memory. MOVCPM can usually be used to do this (MOVCPM is provided with your system as a CP/M utility).

In this example, the size of the CP/M system which this version of ZCPR3 was built on is the same as the size of the ZCPR3 which is being built. Both have the same size of Transient Program Area (TPA).

The following command pulls the Operating System Image off of the System Tracks:

```
B11>sysgen
SYSGEN VER 2.2
SOURCE DRIVE NAME (OR RETURN TO SKIP)a
SOURCE ON A, THEN TYPE RETURN
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

B11>save 45 cpm.bin
```

We now have a file named CPM.BIN on disk. This is the Operating System Image. Note that the "45" above is a number unique to my system and it may be different for the installer's system.

Patching the CP/M System Image

The following illustrates the technique of patching the CP/M Operating System Image. The CCP is replaced by ZCPR3 (ZCPR3.HEX) and the BIOS is replaced by CBIOS (CBIOS.HEX).

In this example, the CP/M Operating System Image begins at 1100H. This may be different on the system being installed.

```
B11>ddt cpm.bin
DDT VERS 2.0
NEXT  PC
2E00 0100
```

The following command displays the CCP area. Your display will probably not be the same as this (I was running ZCPR3 at the time this display was created), but the installer should recognize the two opening JMP instructions as a key to being sure that the CCP is indeed at this location.

```
-d1100 111f
1100 C3 3E C0 C3 3E C0 43 4F 4D 01 24 24 24 20 20 20 .>..>.COM.$$$
1110 20 20 53 55 42 00 00 00 00 00 00 00 00 00 00 00   SUB.......
```

The following zeroes out the CCP area (I prefer to do this—just in case).

```
 -f1100 18ff 0
```

Now the ZCPR3 Command Processor Replacement is read in on top of the CCP area. The offset here is 5100, and it may be different for the system being installed. After the read-in, a dump is done to make sure the image was loaded properly.

```
-izcpr3.hex
-r5100
NEXT  PC
2E00 0000
-d1100 111f
1100 C3 3E C0 C3 3E C0 43 4F 4D 01 24 24 24 20 20 20 .>..>.COM.$$$
1110 20 20 53 55 42 00 00 00 00 00 00 00 00 00 00 00   SUB.........
```

The following command displays the BIOS area. Your display will probably not be the same as this, but the installer should recognize the two opening JMP instructions as a key to being sure that the BIOS is indeed at this location.

```
-d2700 270f
2700 C3 CC D6 C3 70 D7 C3 0C EC C3 39 D6 C3 3F D6 C3 ....p.....9..?..
```

The following zeroes out the BIOS area (I prefer to do this—just in case).

```
 -f2700 2dff 0
```

Now the CBIOS is read in on top of the BIOS area. The offset here is 5100, and it may be different for the system being installed. After the read-in, a dump is done to make sure the image was loaded properly.

```
-icbiosz.hex
-r5100
NEXT  PC
2E00 0000
-d2700 274f
2700 C3 CC D6 C3 70 D7 C3 0C EC C3 39 D6 C3 3F D6 C3 ....p.....9..?..
2710 15 EC C3 18 EC C3 1B EC C3 C3 D7 C3 04 D8 C3 C5 ................
2720 D7 C3 B7 D7 C3 BD D7 C3 F6 D8 C3 EF D8 C3 1E EC ................
2730 C3 CA D7 C3 1B FC C3 21 EC CD 64 D9 C3 0F EC C5 .......!..d.....
2740 CD 64 D9 C1 C3 12 EC CD 0C EC B7 C4 0F EC 21 80 .d............!.
-^C
```

The following SAVE places the new ZCPR3 Operating System Image on disk.

```
 B11>save 46 zcpr3.bin
```

Just to double-check, the image is reloaded and the end of the CBIOS is displayed to make sure the image is complete.

```
B11>ddt zcpr3.bin
DDT VERS 2.0
NEXT PC
2E00 0100
-d2dc0 2dff
2DC0 00 00 00 00 00 00 F8 E0 00 00 03 E3 B8 E2 00 00 ................
2DD0 00 00 00 00 00 00 F8 E0 00 00 8E E3 43 E3 45 6E ............C.En
2DE0 64 20 6F 66 20 43 42 49 4F 53 5A 00 00 00 00 00 d of CBIOSZ.....
2DF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
-^C
```

Placing the Operating System Image

The following now places the Operating System Image onto the system tracks of the desired disks. Note that this image is memory-resident from the previous execution of DDT. Write out the image on as many test disks as desired.

```
B11>sysgen
SYSGEN VER 2.2
SOURCE DRIVE NAME (OR RETURN TO SKIP)
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B...
```

20 Step 7 : System Segment Installation

The selection of the configuration options for the various System Segments of ZCPR3 is described here in some detail.

Resident Command Packages

The following is a reformatted duplicate of the body of a SYSRCP.LIB file. It is provided here to present additional information on how to set the equates. It may be useful to the installer to have this installation manual open to these pages while he is editing this file.

Each entry for the resident commands mentions their transient program counterparts. These utilities usually provide capabilities which exceed those of the programs in the Resident Command Package, but the tradeoff is that each utility program is a separate file on disk which usually occupies more disk space than an entire RCP. In essence, the RCP commands provide quick, convenient capabilities to the user, and the transient utilities provide much greater flexibility and utility to the user. In most reasonable ZCPR3 Systems, both facilities are available.

```
SYSTEM SEGMENT:  SYS1.RCP
SYSTEM:  ZCPR3
WRITTEN BY:  RICHARD CONN

PROGRAM HEADER:  SYSRCP.LIB
AUTHOR:  RICHARD CONN
```

This program header selects the commands to be incorporated into SYS.RCP. It also allows selection of some options for these commands.

IDENTIFICATION

The following ID is a single character, displayed as a part of the RCP ID, which distinguishes this RCP from others made from the same base file (SYSRCP.ASM).

```
RCPID           EQU       'A'
```

With the potential of several RCPs being generated from this one file, RCPID is useful in identifying which RCP is currently loaded to the user. The H command, built into every RCP, prints out the version number of the RCP, including the RCPID character, as well as the names of the commands contained within the RCP.

CP Command

TRANSIENT COUNTERPART: MCOPY

The following equate determines if the CP command is made available. Setting this equate to TRUE enables the CP command.

The CP command copies one file from one DU to another or into the same DU under a different name.

```
    CPON            EQU       TRUE
```

DIR Command

TRANSIENT COUNTERPART: DIR, XD, XDIR

The following equate determines if the DIR command is made available. Setting this equate to TRUE enables the DIR command.

The DIR command displays the directory of files in alphabetical order across the lines to the user.

```
DIRON           EQU       FALSE
```

The DIR command allows two options. One is a flag to tell it to look at both System and Non-System files, and the other is a flag to tell it to look only at System files. By default, DIR looks at Non-System files.

SYSFLG defines the character used to instruct DIR to look at both System and Non-System files. The recommended value is 'A' for All.

SOFLG defines the character used to instruct DIR to look at only System files. The recommended value is 'S' for System.

```
SYSFLG          EQU       'A'
SOFLG           EQU       'S'
```

The following equate determines if the directory displays are sorted by filename and filetype or by filetype and filename. Set SORTNT to TRUE to sort by name and type, FALSE to sort by type and name.

```
SORTNT          EQU       TRUE
```

The following equates define some features of the directory display. If WIDE is TRUE, the file names are spaced farther abort; if WIDE is FALSE, they are closer together (for a 64-column display). FENCE defines the character used to separate the file name entries in the display.

```
WIDE            EQU       TRUE
FENCE           EQU       '|'
```

ERA Command

TRANSIENT COUNTERPART: ERASE

The following equate determines if the ERA command is made available. Setting this equate to TRUE enables the ERA command. The ERA command erases files.

```
ERAON           EQU       TRUE
```

LIST and TYPE Commands

TRANSIENT COUNTERPART: PRINT and PAGE

The following equate determines if the LIST and TYPE commands are made available. Setting this equate to TRUE enables these commands.

The LISTON equate can disable the LIST command without affecting the TYPE command.

The TYPE command displays a group of files on the CRT while the LIST command prints a group of files on the Printer.

```
LTON        EQU     TRUE
LISTON      EQU     TRUE
```

TYPE can be made to page or not page by default. If PGDFLT is TRUE, TYPE pages by default and does not page if the PGFLG character (recommended to be 'P') is used. If PGDFLT is FALSE, TYPE pages only when the PGDFLG character is seen in the command line.

```
PGDFLT      EQU     TRUE
PGDFLG      EQU     'P'
```

NLINES defines the number of lines on the user's CRT screen. This is usually 24.

```
NLINES      EQU     24
```

PEEK and POKE Commands

TRANSIENT COUNTERPART: None (Subset of DDT)

The following equates determine if the PEEK and POKE commands are made available. Setting these equates to TRUE enables these commands.

The PEEK command allows the user to examine a chunk of memory.

```
PEEKON      EQU     TRUE
POKEON      EQU     TRUE
```

PROT Command

TRANSIENT COUNTERPART: PROTECT

The following equate determines if the PROT command is made available. Setting this equate to TRUE enables the PROT command.

The PROT command sets the file protection attributes for a group of files.

```
PROTON      EQU     TRUE
```

REN Command

TRANSIENT COUNTERPART: RENAME

The following equate determines if the REN command is made available. Setting this equate to TRUE enables the REN command. The REN command changes the name of one file to another.

```
RENON           EQU       TRUE
```

REG Command

TRANSIENT COUNTERPART: REG

The following equate determines if the REG command is made available. Setting this equate to TRUE enables the REG command.

```
REGON           EQU       FALSE
```

WHL Command

TRANSIENT COUNTERPART: WHEEL

The following equate determines if the WHL command is made available. Setting this equate to TRUE enables the WHL command.

The WHL command is used to turn off the Wheel Byte (make the user non-priveleged) or to turn on the Wheel Byte (make the user priveleged).

Also, this equate enables the WHLQ command, which displays the state of the Wheel Byte.

```
WHLON           EQU       FALSE
```

The following equate defines the password to be used by the WHL command. It must always be 8 bytes long (trailing spaces allowed) and must be upper-case.

```
WPASS   MACRO
        DB      'SYSTEM  '      ;8 characters
        ENDM
```

The Wheel equate table enables the WHEEL facility of ZCPR3. With this facility, a WHEEL BYTE, which exists somewhere in memory, is examined before a set of installer-selected commands are executed. If this byte is not zero, then the command proceeds. If it is zero, then the command is not allowed to proceed and is exited with an error message.

The following set of equates make each of the indicated commands selectable to respond to the Wheel Byte or not. For instance, if WERA=TRUE, then it responds to the Wheel Byte; if WERA=FALSE, it does not.

These options will be effective only if a Wheel Byte is Defined (Z3WHL NE 0)

```
WCP        equ     FALSE     ;Make CP    a Wheel-Oriented Command
WDIR       equu    FALSE     ; "   DIR   "  "             "         "
WERA       equ     FALSE     ; "   ERA   "  "             "         "
WLIST      equ     FALSE     ; "   LIST  "  "             "         "
WPEEK      equ     FALSE     ; "   PEEK  "  "             "         "
WPOKE      equ     FALSE     ; "   POKE  "  "             "         "
WPROT      equ     FALSE     ; "   PROT  "  "             "         "
WREG       equ     FALSE     ; "   REG   "  "             "         "
WREN       equ     FALSE     ; "   REN   "  "             "         "
WTYPE      equ     FALSE     ; "   TYPE  "  "             "         "

WHEEL      set     WCP OR WDIR OR WERA OR WLIST OR WPEEK OR WPOKE
WHEEL      set     WHEEL OR WPROT OR WREG OR WREN OR WTYPE
```

NOTE Command

TRANSIENT COUNTERPART: NOTE

NOTE is simply a NOP (do nothing) command which can be used to place comments into multiple command lines. Setting the following equate to TRUE enables the NOTE Command.

```
NOTEON          EQU        TRUE
```

The NOTE command is very convenient in the creation of commented displays and command files. It is generally recommended to implement this command as a resident command within the ZCPR3 Command Processor itself rather than within an RCP since the ZCPR3 Command Processors tend to have more room to spare than RCPs and it is frequently desirable to save as much space within an RCP as possible.

ECHO Command

TRANSIENT COUNTERPART: ECHO

The following equate enables the ECHO command.

ECHO is useful in issuing both messages (to the user, say within a command file during execution) and escape sequences. ECHO can send its output to the console (by default) or to the printer (if the first non-blank character is a dollar sign). It uses BIOS calls, so all control characters are passed exactly. Hence, console-level programming of such devices (CRTs and Printers) is possible.

The ECHOLST equate determines if ECHO is allowed to direct its output to the printer. If ECHOLST is TRUE, ECHO may direct its output to the printer via the $ prefix character in the text.

```
ECHOON          EQU       TRUE
ECHOLST         EQU       TRUE
```

The ECHO transient is not very large, and it is frequently more convenient to have ECHO implemented in an RCP. However, since space within RCPs is frequently at a premium, it may be necessary to employ the ECHO transient.

Flow Command Packages

The following is a reformatted duplicate of the body of a SYSFCP.LIB file. It is provided here to present additional information on how to set the equates. It may be useful to the installer to have this installation manual open to these pages while he is editing this file.

A key decision to be made in the creation of FCPs is whether to implement the IF command as a COM file or within the FCP itself. The following tradeoff should be considered:

1. As a COM file, the IF command offers many more options and flexibility for condition processing than an FCP-resident IF.
2. As a COM file, the IF command adds overhead by having to be located and loaded from disk and then executed.

In the following text, we also show the equates that set options for an FCP-resident IF command. IF.COM contains all of these options and more. Refer to the associated HLP file for more detail.

```
SYSTEM SEGMENT:  SYS1.FCP
SYSTEM:  ZCPR3
CUSTOMIZED BY:  RICHARD CONN

PROGRAM HEADER:  SYSFCP.LIB
AUTHOR:  RICHARD CONN
```

This program header defines the IF Conditions to be placed into the target SYS.FCP file (generated by assembling SYSFCP.ASM).

IF Negation

The following equate determines if leading negation is to be allowed. If this equate is TRUE, then forms like the following are permitted:

```
IF ~EXIST filename.typ
```

These forms complement the meaning of the test (the above returns TRUE if filename.typ does NOT exist).

```
IFONEG          EQU       TRUE
```

Assuming IFONEG to be TRUE, the following equate defines the character to be placed in front of the IF option to indicate that negation is to be performed. In the above example, this character was tilde (~).

```
NEGCHAR         EQU       '~'
```

IF: T (True) or F (False)
Setting the following equate to TRUE enables the simple T and F options to IF. The format of this option is:

IF T or IF F

and it always returns TRUE or FALSE, resp.

```
IFOTRUE      EQU      FALSE
```

IF: EM (Empty)
Setting the following equate to TRUE enables IF to test to check whether or not the indicated file is empty. The test returns TRUE if the indicated file does not exist or is empty.

```
IFOEMPTY     EQU      FALSE
```

IF: ER (Error)
Setting the following equate to TRUE enables IF to test the error code byte (program error code byte). If this byte is 0 (no error), it returns TRUE, else it returns FALSE.

```
IFOERROR     EQU      TRUE
```

IF: EX (Exist)
Setting the following equate to TRUE enables IF to test for the existence of a file. The test returns TRUE if the indicated file exists.

```
IFOEXIST     EQU      TRUE
```

IF: IN (Input)
Setting the following equate to TRUE enables user input of the character T (or any other character for FALSE). ZEX processing is suspended for this single-character input.

```
IFOINPUT     EQU      TRUE
```

IF: NU (Null)
Setting the following equate to TRUE enables IF to test to see if the second argument which follows is NULL (not specified) or not. This test is particularly useful in command file processing to see if, for example, argument $2 exists and to include it if it does.

```
IFONULL      EQU      TRUE
```

IF: n (Register Value)
Setting the following equate to TRUE enables IF to test to see if the indicated register contains the indicated value. If this is preceded by the NEGCHAR and IFONEG is TRUE, then the test checks to see that the indicated register does not contain the indicated value.

```
IFOREG          EQU        TRUE
```

IF: WH (Wheel)
Setting the following equate to TRUE enables IF to check whether the Wheel Byte is set or not. If the Wheel Byte is set, the statement IF WHEEL returns TRUE.

```
IFOWHEEL        EQU        FALSE
```

IF: TC (TCAP)
Setting the following equate to TRUE enables IF to test to see if the ZCPR3 TCAP contains a terminal definition or not. This test is particularly useful in command file or alias processing to check whether a Z3TCAP entry is defined, and to invoke screen-oriented routines if it is.

```
IFOTCAP         EQU        FALSE
```

IF: fcb1=fcb2
Setting this equate to TRUE will enable IF to check whether the two FCBs contain the same values. If so, the IF returns TRUE; if not, the IF returns FALSE. Enabling this equate eliminates the need for the NULL test, since a test for null can be performed by using the syntax: IF fcb1=

```
IFOEQ           EQU        TRUE
```

COMIF - Run IF.COM
Setting this equate to TRUE will cause an IF statement executed during an IF TRUE or NO IF state to look in the ROOT directory for the file IF.COM, and, if found, load IF.COM and transfer control to it. If IF.COM is not found, then IF F is raised. Using IF.COM provides much more power and flexibility but also requires IF.COM to be present and takes up disk space.

```
COMIF           EQU        FALSE
```

NOISE—Have FCP Print IF Status
Setting this equate to TRUE will cause any change in the IF status to be displayed to the user. This is useful for debugging purposes, but in normal runs, particularly where ALIASes are concerned, it is usually desirable to reduce the "noise" by setting this equate to FALSE.

```
NOISE           EQU        FALSE
```

Input/Output Packages
Input/Output Packages are very machine-specific but, like all packages, they provide a machine-independent interface to the ZCPR3 System in their visible sections. The hidden part performs the actual implementation of the routines. Like the structure of the BIOS, the visible section of an I/O Package consists of a JMP table.

The installer who is interested in incorporating Input/Output Packages into the system he is installing is referred to the source code file SYSIOP.ASM. This file can be used as a template through which to create other I/O Packages. It is filled with comments outlining the functions being performed, and I feel that this should be adequate.

Named Directory Files

The following is a reformatted duplicate of the body of a SYSNDR.LIB file. It is provided here to present additional information on how to set the equates. It may be useful to the installer to keep the book open to these pages while he is editing this file.

```
DATA FILE:  SYSNDR.LIB
AUTHOR:  Richard Conn
VERSION:  1.0
DATE:  24 Feb 84
```

SYSNDR.LIB defines the structure of the memory-based named directory. It also defines a few elements for it and is suitable for enclosure in an NDR file. The general structure is:

```
            DB      Disk,User       ; A=1
            DB      'NDIRNAME'      ; 8 chars
            DB      'PASSWORD'      ; 8 chars
            ...                     ; other entries
            DB      0               ; End of NDR

defdu       macro   ?disk,?user
            db      ?disk-'@'                   ; Convert Disk
            db      ?user                       ; User is OK
            endm
```

The entire file is implemented as one macro (which follows). The SYSNDR.ASM file simply refers to this macro and expands it.

The named directories shown below are recommended standards. In time, there will be utilities which base a part of their operations on these names.

```
sysndr     macro
```

The BASE directory is a working scratch area on the first disk.

```
defdu   'A',0
db      'BASE    '
db      '        '
```

The ROOT directory is the last directory referenced in the Command Search Path. This is where all the general purpose COM files are located.

```
defdu   'A',15
db      'Root    '
db      '        '
```

The HELP directory is where the online documentation files are stored.

```
        defdu   'A',16
        db      'HELP'   '
        db      '        '
```

The BACKUP directory is where files are copied to (by default) for backup purposes.

```
        defdu   'C',0
        db      'BACKUP  '
        db      '        '

        db      0                   ;End of List
        endm
```

TCAP Files

The programs TSELECT and TCMAKE are used to create the *.Z3T files which are loaded by the LDR.COM utility. The loaded file establishes the characteristics of the user's CRT terminal, and this information is used by screen-oriented utilities, such as SHOW, to perform their functions.

Environment Descriptor

The following is a reformatted duplicate of the body of a SYSENV.LIB file. It is provided here to present additional information on how to set the equates. It may be useful to the installer to have this installation manual open to these pages while he is editing this file.

The entire file is one macro which is referenced by SYSENV.ASM. SYSENV inserts a JMP 0 instruction in front of this macro to complete the structure of the SYS.ENV file.

```
    LIBRARY:  SYSENV.LIB
    AUTHOR:  Richard Conn
    Version:  1.0
    Date:  18 May 84
    Previous Versions:  None
```

SYSENV is the definition for my ZCPR3 environment.

```
sysenv      macro
;
;  Environment Descriptor
;   If inline, there is a leading JMP just before this
;
envorg1:
    db      'Z3ENV'          ; Environment ID
    db      1                ; class 1 environment (external)
```

A Class 1 environment is external to the utility using it. This type of Environment Descriptor is located at a buffer somewhere in memory, and the ZCPR3 utilities simply contain a 2-byte pointer to its address. A Class 2 environment is internal to the utility using it. This type of Environment Descriptor is located within the utility itself, taking up 256 bytes. It is recommended that the ZCPR3 System be configured using an external Environment Descriptor.

The following addresses and values are extracted from Z3BASE.LIB.

```
dw      expath          ; external path address
db      expaths         ; number of 2-byte elements in path

dw      rcp             ; RCP address
db      rcps            ; number of 128-byte blocks in RCP

dw      iop             ; IOP address
db      iops            ; number of 128-byte blocks in IOP

dw      fcp             ; FCP address
db      fcps            ; number of 128-byte blocks in FCP

dw      z3ndir          ; NDR address
db      z3ndirs         ; number of 18-byte entries in NDR

dw      z3cl            ; ZCPR3 Command Line
db      z3cls           ; number of bytes in Command Line

dw      z3env           ; ZCPR3 Environment Descriptor
db      z3envs          ; number of 128-byte blocks

dw      shstk           ; Shell Stack address
db      shstks          ; number of shsize-byte entires
db      shsize          ; size of a Shell Stack entry

dw      z3msg           ; ZCPR3 Message buffer

dw      extfcb          ; ZCPR3 External FCB

dw      extstk          ; ZCPR3 External Stack
```

The following flag is used by some ZCPR3 System utilities to determine how verbose they are in providing messages and information to the user. The QUIET.COM utility can be used to change this flag dynamically.

```
db      0               ; quiet flag (1=quiet, 0=not quiet)

dw      z3whl           ; address of Wheel Byte
```

This data value is used by the timing routines.

```
db      4               ; Processor Speed in MHz
```

The following values should correspond to those selected in the Z3HDR.LIB file.

```
db      'D'-'@'         ; maximum disk
db      31              ; maximum user
```

The following value is used to instruct the utilities as to whether they should accept the DU form or not. If disabled (set to 0), the only way to reference a directory is with the DIR (named) form, and password protection is directly provided by this.

```
db      1               ; 1=OK to accept DU, 0=not OK
```

Some ZCPR3 utilities, such as PRINT and PAGE, draw information from these buffers to determine several key attributes of the devices they are dealing with. The CPSEL utility can be used to dynamically change the CRT and Printer selections.

```
db      0               ; CRT selection (0=CRT 0, 1=CRT 1)
db      0               ; Printer selection (n=Printer n)

db      80              ; width of CRT 0
db      24              ; number of lines on CRT 0
db      22              ; number of lines of text on CRT 0

db      132             ; width of CRT 1
db      24              ; number of lines on CRT 1
db      22              ; number of lines of text on CRT 1

db      80              ; width of Printer 0
db      66              ; number of lines on Printer 0
db      58              ; number of lines of text on Printer 0
db      1               ; form feed flag (0=can't formfeed, 1=can

db      102             ; width of Printer 1
db      66              ; number of lines on Printer 1
db      58              ; number of lines of text on Printer 1
db      1               ; form feed flag (0=can't formfeed, 1=can
db      80              ; width of Printer 2
db      66              ; number of lines on Printer 2
db      58              ; number of lines of text on Printer 2
db      0               ; form feed flag (0=can't formfeed, 1=can)

db      102             ; width of Printer 3
db      66              ; number of lines on Printer 3
db      58              ; number of lines of text on Printer 3
db      0               ; form feed flag (0=can't formfeed, 1=can)
```

The ZCPR3 shell named SH can deal with symbols (variables) which are assigned text strings as values. This buffer defines the name of the file which programs like SH refer to in order to resolve variable references. As many shell variable files as desired may be available in this fashion.

```
db      'SH      '      ; shell variable filename
db      'VAR'           ; shell variable filetype
```

These buffers are available to store file names and other data which are passed from one utility to another which is executed later. In general, entries 3 and 4 are available to the ZCPR3 utility programmer as general-purpose buffers. Entries 1 and 2 are used by some ZCPR3 System utilities at this time.

```
db      '        '      ; filename 1
db      '   '           ; filetype 1

db      '        '      ; filename 2
db      '   '           ; filetype 2

db      '        '      ; filename 3
db      '   '           ; filetype 3

db      '        '      ; filename 4
db      '   '           ; filetype 4

ds     80H-($-envorg1+3)       ; make exactly 80H bytes long
                               ; (+3 compensates for leading JMP)
```

The following is the TCAP entry for the TVI 950. If LDR.COM loads a *.Z3T file, this buffer will be overlaid (if the Environment Descriptor is External).

```
;
; Terminal Capabilities Data
;
```

```
envorg2:
    DB      'TVI 950                  ;Name of Terminal
    DB      'K'-'@'                   ;Cursor UP
    DB      'V'-'@'                   ;Cursor DOWN
    DB      'L'-'@'                   ;Cursor RIGHT
    DB      'H'-'@'                   ;Cursor LEFT
    DB      00                        ;CL Delay
    DB      00                        ;CM Delay
    DB      00                        ;CE Delay
    DB      1bh,'*',0                 ;CL String
    DB      1bh,'=%+ %+ ',0           ;CM String
    DB      1bh,'t',0                 ;CE String
    DB      1bh,')',0                 ;SO String
    DB      1bh,'(',0                 ;SE String
    DB      0                         ;TI String
    DB      0                         ;TE String

    ds      80H-($-envorg2)           ; make exactly 80H bytes lon

;
;  End of Environment Descriptor
;
    endm
```

21 Step 8 : Utility Installation

The Z3INS Utility

The Z3INS utility is designed to make the ZCPR3 utility installation process simple. All files to be installed must be in the current directory when Z3INS is executed. A *.ENV file for the target system and an installation file (*.INS) containing the names of the programs to be installed must also be in the current directory.

Z3INS reads in an Environment Descriptor file (*.ENV) and an Installation File (*.INS). It then looks for lines in the file containing file names (one name per line) and loads the indicated files, trying to install them with the Environment Descriptor information. Z3INS is invoked by a command line of the following form:

```
    Z3INS mysys.ENV myinstal.INS
```

A ZCPR3 installation file is a text file containing two types of lines: a comment line, which begins with a semicolon (;), and a line containing an unambiguous file name (leading spaces are not significant), which is a file to be installed. For example:

```
; This is an installation file for my new utilities
; UTIL1.COM and UTIL2.COM are going to be installed --
 util1.com
     util2.com
; UTIL3 is really neat
util3.com
```

Case is not significant. Leading spaces on each line are ignored. Any file name *must* be unambiguous. Listing 21-1, which follows, shows a sample session in which Z3INS installs a system, using a previously prepared *.INS file. Listing lines that begin with a semicolon are explanatory comments and are not displayed on the console.

Sample Session

```
B1:ASM>z3ins sys.env zcpr3.ins
Z3INS  Version 1.0
;
;  Installation Begins --
;

           << Detail Left Out >>

;
;  Set 1
;
** Installing File ALIAS    .COM
** Installing File CD       .COM
** Installing File CMDRUN   .COM
** Installing File COMMENT .COM
** Installing File CPSEL   .COM
** Installing File CRC     .COM
** Installing File DEV     .COM
** Installing File DEVICE  .COM
** Installing File DIFF    .COM
** Installing File DIR     .COM
** Installing File ECHO    .COM
** Installing File ERASE   .COM
;
;  Set 2: Error Handlers
;
** Installing File ERROR1  .COM
** Installing File ERROR2  .COM
** Installing File ERROR3  .COM
** Installing File ERROR4  .COM
** Installing File ERRORX  .COM
** Installing File SHOW    .COM
```

<< Detail Left Out >>

```
;
;  NOTE does not install because it is so small and really does
;  not need to know about ZCPR3
;
;note.com
;
;  Set 9:  Z3INS
;
** Installing File Z3INS   .COM
;
; End of ZCPR3 Installation
;
** Installation Complete **
```

Listing 21-1. Sample Run of Z3INS

Assembling Distribution Files

The following files require their specialized command files in order to be assembled. If the installer is installing the system for the first time and wishes to assemble these utilities, he may have to follow the steps outlined in the command files in order to perform the assemblies.

Utility	*Command File Required*
ALIAS.COM	ALIAS.ZEX
ZEX.COM	ZEX.ZEX

The following files in the Phase I distribution are assembled by the command lines (assuming that Z3LIB.REL and SYSLIB.REL are in the current directory and that $1 is the file):

```
M80 =$1
L80 /P:100,$1,Z3LIB/S,SYSLIB/S,$1/N,/U,/E
```

Files:

```
CD          CMDRUN      COMMENT     CPSEL
CRC         DEV         DEVICE      DIFF
DIR         ECHO        ERASE       ERROR1
ERROR3      ERROR4      ERRORX      FINDF
GOTO        HELPCK      IF          IFSTAT
LDR         MCOPY       MENUCK      MKDIR
NOTE        PAGE        PATH        PROTECT
PWD         QUIET       RECORD      REG
RENAME      SAK         SETFILE     SH
SHCTRL      SHDEFINE    SHFILE      SHVAR
SUB         TCCHECK     TCMAKE      TCSELECT
UNERASE     WHEEL       XD          XDIR
Z3INS       Z3LOC
```

The following files in the Phase I distribution are assembled by the command lines (assuming that VLIB.REL, Z3LIB.REL, and SYSLIB.REL are in the current directory):

```
M80 =$1
L80 /P:100,$1,VLIB/S,Z3LIB/S,SYSLIB/S,$1/N,/U,/E
```

Files:

```
ERROR2      HELP        MENU        SHOW
```

The following files in the Phase I distribution are assembled by the command lines (assuming that Z3LIB.REL and SYSLIB.REL are in the current directory) if the TIME option is enabled:

```
M80 =$1
L80 /P:100,$1,TIMELIB/S,Z3LIB/S,SYSLIB/S,$1/N,/U,/E
```

Files:

```
HELPPR          PRINT
```

Files distributed in Phase II will be provided with associated documentation on their assembly procedures.

22 TCAP Facility

ZCPR3 Terminal Capabilities (TCAP)

The ZCPR3 Terminal Capabilities (TCAP) Facility is an integral part of the ZCPR3 System. By means of the TCAP Facility, the user's terminal is defined to ZCPR3 in such a way that programs in the ZCPR3 System can perform a variety of screen-oriented functions with the user's terminal. The TCAP Facility is fundamental to ZCPR3, and it is a part of the ZCPR3 Environment Descriptor.

The TCAP entries contain the following information on their respective terminals:

o initialization/deinitialization sequences
o characters generated by the arrow keys
o sequence for clearing the screen
o sequence for positioning the cursor
o sequence for erasing to end of line
o highlight/non-highlight sequences

With this information, programs such as VFILER, VMENU, and HELP can perform their functions with a much higher degree of "flash" and user-friendliness than they would otherwise. By simply loading the TCAP entry for another terminal into the environment descriptor, all ZCPR3 programs are automatically reconfigured for the new terminal and can continue to function without modification.

Two utilities are provided to assist the user in the creation of Z3T files for his terminals. TCSELECT allows the user to select a predefined terminal from the Z3TCAP file, and TCMAKE allows the user to define a terminal which is not covered by the Z3TCAP file.

Most of the information in this chapter provides details on the structure of the Z3T files and gives the TCMAKE user enough detail to define his terminal. Providing this information is the main purpose of this chapter.

The file Z3TCAP contains information on over 40 terminals. The TCSELECT program prints a number of menus containing the names of the terminals in this file and allows the user to select one, storing its information either directly into the memory-resident Environment Descriptor or into a file of type Z3T which may later be loaded by the LDR utility.

If the user's terminal is not already defined in the Z3TCAP file, the TCMAKE program is used to define his terminal. TCMAKE allows the user to interactively define each of the key attributes of his terminal and create a file of type Z3T when done. This file may later be loaded by the LDR utility.

Internal Structure of a Z3T File

A Z3T File defines the characteristics of a particular terminal. Each Z3T file contains the following information:

o the name of the terminal
o the codes generated by the arrow keys
o the byte sequences required:
  to clear the screen
  to position the cursor
  to clear to end of line

to highlight chars
to initialize and deinitialize the terminal

The following is the exact structure of a Z3T file:

```
name:
     DS   16    ; Name of Terminal
arrows:
     DS   4     ; Bytes generated by arrow keys
delays:
     DS   3     ; Delays for Screen Clear, Cursor Motion, and
                ;   Clear to End-of-Line

cl:  DS   N1+1 ; Sequence used for Screen Clear
cm:  DS   N2+1 ; Sequence used for Cursor Motion (gotoxy)
ce:  DS   N3+1 ; Sequence used for Clear to End-of-Line
so:  DS   N4+1 ; Sequence used to begin highlighting
se:  DS   N5+1 ; Sequence used to end highlighting
ti:  DS   N6+1 ; Sequence used to initialize terminal
to:  DS   N7+1 ; Sequence used to deinitialize terminal
```

The following defines the TCAP records, which are:

1. the name of the terminal
2. the definition of the arrow keys
3. the delay constants for screen clear, cursor motion, and clear to end-of-line
4. the definition of the screen clear char sequence
5. the defn of the cursor motion char seq
6. the defn of the clear to EOL char seq
7. the defn of the highlight/end-highlight char seq
8. the defn of the init/deinit terminal char seq

Each of these record definitions is similar: the structure of the record (in assembly language terminology), comments on how the record is defined and what values are valid for it. Examples of valid record structures are provided for each record definition.

Terminal Name

Structure:

```
DS   16   ; Name of Terminal (Space Fill on Right)
```

Comment:

The name of the terminal is always 16 bytes long. If the name takes less than 16 bytes, space fill occurs right of the last character.

Examples:

```
DB   'ADDS Consul 980 '
DB    'ADM 2 '
```

Arrow Keys
Structure:

```
DS    1     ; Byte Generated by Cursor UP
DS    1     ; Byte Generated by Cursor DOWN
DS    1     ; Byte Generated by Cursor RIGHT
DS    1     ; Byte Generated by Cursor LEFT
```

Comment:
If your terminal has arrow keys on it *which generate only one byte when depressed*, then these keys may be defined in the Z3T file. When a program calls for the use of arrow keys, it will use the values stored here.

If your terminal does not have arrow keys or has arrow keys which generate more than one byte when depressed, these keys may not be defined in the Z3T file. Zero (0) is stored in all four bytes of the "arrow key" record. In this case, the program will respond to the WordStar (trademark, Micropro) arrow key convention (^E is UP, ^X is DOWN, ^D is RIGHT, and ^S is LEFT):
Examples:

```
DB    'K'-'@'    ; ADM 31 ^K for Cursor UP
DB    'J'-'@'    ;        ^J for Cursor DOWN
DB    'L'-'@'    ;        ^L for Cursor RIGHT
DB    'H'-'@'    ;        ^H for Cursor LEFT

DB    0,0,0,0    ; None for H19 because of 2-char seqs
                 ;   Word Star Convention will be used
```

Function Delays
Structure:

```
DS     1     ; Delay (in mS) after sending clear screen
DS     1     ; Delay (in mS) after sending gotoxy
DS     1     ; Delay (in mS) after sending clear to EOL
```

Comment:
Each of these bytes defines the number of milliseconds a program will delay after sending a particular sequence to the user's terminal. Some terminals require this type of delay. If a sequence requires no delay, the value of zero (0) should be placed in the corresponding byte.
Examples:

```
DB    20,0,0     ; BANTAM 550 - 20mS for Clear Screen, 0
                 ;  for Cursor Motion and Clear to EOL
```

```
DB    0,0,0      ; TVI 950 - No Delays
```

Clear Screen Sequence
Structure:

```
DS    N1    ; Bytes in clear screen sequence
DB    0
```

Comment:
This sequence of bytes, up to but not including the terminating 0, is sent to the user's terminal in order to clear his screen. If it is necessary to include a binary zero in this sequence, the two bytes

```
DB    '\',0
```

will transmit as one binary 0 and

```
DB    '\\'
```

will transmit as one backslash.
In general, a backslash (\) is the quote character, and any byte which follows it is transmitted literally to the user's terminal.
If a terminal requires that trailing nulls follow the last character of the sequence for the purpose of screen settling (rather than using the delay byte), nulls can be appended into the sequence by using the quote character.
Examples:

```
DB   1BH,';',0        ; Clear Screen for ADM2
DB   'L'-'@',0        ; Clear Screen for ADDS Viewpoint
DB   1BH,'?',1BH,'E'-'@',0
                      ; Clear Screen for Concept 108
```

Cursor Motion (GOTOXY) Sequence
Structure:

```
DS   N2    ; Bytes in gotoxy sequence
DB   0
```

Comments:
This sequence of bytes is sent to the user's terminal in order to position the cursor on his screen. The quote character (\) can be used, as in the Cursor Motion sequence, to allow quote characters and nulls to be sent.
Unlike the other sequences in the TCAP records, the Cursor Motion sequences vary depending upon the position on the screen. For instance, to place the cursor at row 4, column 4 (home is row 0, col 0) on a TVI 950, the sequence

```
DB 1BH,'=$$',0
```

is used, but to position at row 6, column 6, the sequence

```
    DB 1BH,'=&&',0
```

is used.

In order to express such variable-value sequences, the ZCPR3 TCAP provides for equations which define how to compute the byte to be output. The TCAP sequence:

```
    DB 1BH,'=%+ %+ ',0
```

defines how to compute the values to be output in order to move the cursor for the TVI 950. The TCAP Cursor Motion sequence

```
    DB 1BH,'=%+ %+ ',0
```

is broken down as follows:

Element	*Meaning*
1BH	Output 1B hex (the ESCAPE char)
'='	Output the character '='
'%+ '	Add ' ' (20H) to the row value and output
'%+ '	Add ' ' (20H) to the column value and output

Cursor Motion Interpreter Commands

The percent character (%) instructs the cursor motion sequence interpreter to look for a command, and it processes the following characters as such. If it is desired to output '%' itself, the sequence '\%' is used.

The commands recognized by the cursor motion command interpreter will now be discussed in detail. These commands are the following (case is not significant):

%R	Reverse order from row/col to col/row
%I	Home position is (1,1) rather than (0,0)
%.	Print current value (row or col) in binary
%2	Print current value as 2 ASCII decimal digits
%3	Print current value as 3 ASCII decimal digits
%d	Print current value as N ASCII decimal digits (no leading zeroes)
%+n	Add n to current value and output in binary
%>xy	Add y to current value if it is greater than x

%R Command

The cursor motion sequence interpreter assumes that the value of the row will be output before the value of the column. If the column is to be first, the command

'%r' or '%R'

instructs the cursor motion sequence interpreter to output the column and then the row. The '%R' command must be present in the sequence before the first value is output, and '%R' acts solely to command the interpreter (no bytes are output by '%R').

%I Command

The cursor motion sequence interpreter also assumes that the value of the home position is row 0, column 0. If it is convenient to set this position to row 1, column 1, the command

'%i' or '%I'

is used. Like '%R', '%I' must be used before the first value is output.

The TVI 950 can be defined in two ways:

```
DB    1BH,'=%+ %+ ',0

              or

DB    1BH,'%i=%+',1FH,'%+',1FH,0
```

Output Commands

The rest of the cursor motion sequence interpreter commands deal with the format of the output. They allow the following types of outputs:

%.	binary value (^A is output as 1)
%2	2 ASCII Decimal Digits (^A is output as '01')
%3	3 ASCII Decimal Digits (^A is output as '001')
%d	As many ASCII Decimal Digits as needed (^A as '1')
%+n	Add offset ('%+ ' outputs ^A as 1+' ' or '!')
%>xy	Add offset if limit reached ('%> 1 outputs ^A as 1 and !'as")"

Examples:

```
DB    1BH,'Y%+ %+ ',0        ; ADDS Viewpoint

      1BH   = output 1BH (ESCAPE char)
      'Y'   = output char 'Y'
      '%+ ' = output row + ' ' (20H) in binary
      '%+ ' = output col + ' ' (20H) in binary

DB    1BH,'[%d;%dH',0        ; H19 (ANSI Mode)

      1BH   = output 1BH (ESCAPE char)
      '['   = output char '['
      '%d'  = output row as ASCII decimal digits
      ';'   = output char ';'
      '%d'  = output col as ASCII decimal digits
      'H'   = output char 'H'
```

Clear to End of Line

Structure:

```
DS   N3 ; Bytes in clear to end of line sequence
DB   0
```

Comments:
The Clear to End of Line sequence is used to clear the line starting at the cursor position to the end of the screen. Only this part of the current line is cleared. The rules for specifying this sequence are the same as those for Screen Clear.
Example:

```
DB   1BH,'T',0       ; ADM 2
```

Begin and End Highlighting
Sequences:

```
DS   N4   ; Bytes in sequence to begin highlighting
DB   0

DS   N5   ; Bytes in sequence to end highlighting
DB   0
```

Comments:
The "begin highlighting" sequence is used to begin highlight mode on the user's terminal. This may be reverse video, dim, or some other non-standard method for displaying characters on the screen. In order for a terminal to support this feature, the following must be true:

1. Issuing this sequence must NOT change the position of the cursor on the screen.
2. Characters highlighted must be output in exactly the same way non-highlighted characters are (eg, setting the MSB of the highlighted chars is not allowed).

These sequences are always used as follows:

1. the BEGIN HIGHLIGHT sequence is output
2. a set of characters to highlight is output
3. the END HIGHLIGHT sequence is output

The rules for specifying these sequences are the same as those for Screen Clear.
Example:

```
DB   'N'-'@',0       ;ADDS Viewpoint
```

Terminal Init and Deinit
Sequences:

```
DS   N6   ; Bytes in sequence to init terminal
DB   0
```

```
DS    N7    ; Bytes in sequence to deinit terminal
DB    0
```

Comments:

Before any video routines are executed, the terminal initialization sequence is sent to the terminal. After the use of the terminal is completed by a program, the deinitialization sequence is sent. The rules for specifying these sequence are the same as those for Screen Clear.

Terminal Control Sequences 1 (General)

The structure of most TCAP control sequences is:

```
DS    N    ; Bytes in sequence
DB    0
```

This sequence of bytes, up to but not including the terminating 0, is sent to the user's terminal in order to perform some function. If it is necessary to include a binary zero in this sequence, the two bytes

```
DB    '\',0
```

will transmit as one binary 0 and

```
DB    '\\'
```

will transmit as one backslash.

A backslash (\) is the quote character, and any byte which follows it is transmitted literally to the user's terminal.

If a terminal requires that trailing nulls follow the last character of the sequence for the purpose of screen settling (rather than using the delay byte), nulls can be appended to the sequence by using the quote character.

Cursor Motion sequences follow these rules with the addition that the character "%" prefixes a cursor motion interpreter command. If it is desired to simply output this character in a cursor motion sequence, the quote character can be used:

```
DB    '\%'
```

Terminal Control Sequences 2 (Cursor Motion)

Cursor Motion sequences are different from the other sequences defined in the TCAP, in that cursor motion sequences contain embedded commands for the cursor motion interpreter. All Cursor Motion sequences are of the following general format:

<prefix sequence> <commands> <infix seq> <cmd> <postfix seq>

For example, the DEC VT100 terminal uses the following sequence for cursor motion:

```
DB    1BH,'[%i%d;%dH',0
```

where:

```
1BH,'['    = prefix chars 1BH (ESCAPE) and '['
%i         = command: home is 1,1
%d         = command: output row as ASCII dec chars
';'        = infix char ';'
%d         = command: output col as ASCII dec chars
'H'        = suffix char 'H'
```

The prefix, infix, and postfix sequences are optional, and only the commands to output the row and column are required in any cursor motion sequence definition.

Cursor Motion is the only required entry in a TCAP for a terminal. All other sequences may be empty (null), and the lack of these other sequences will be compensated for. Cursor motion, however, cannot easily be simulated, and is therefore mandatory.

The following table summarizes all of the Cursor Motion interpreter commands.

Command	*Output Format*
%.	Binary Value
%2	2 ASCII Decimal Digit Chars ('23')
%3	3 ASCII Decimal Digit Chars ('123')
%d	As many ASCII Decimal Digit Chars as needed
%+n	Add the value of the byte following the '+' and output in binary
%>xy	If value > x, output value+y in binary; else output value in binary

Command	*Cursor Motion Interpreter Action*
%i	Set Home to 1,1 (default is 0,0)
%r	Output Col, then Row (default is Row, then Col)

Examples:

```
DB    1BH,'Y%+ %+ ',0       ; ADDS Viewpoint

      1BH    = output 1BH (ESCAPE char)
      'Y'    = output char 'Y'
      '%+ ' = output row + ' ' (20H) in binary
      '%+ ' = output col + ' ' (20H) in binary

DB    1BH,'[%d;%dH',0       ; H19 (ANSI Mode)

      1BH    = output 1BH (ESCAPE char)
      '['    = output char '['
      '%d'   = output row as ASCII decimal digits
      ';'    = output char ';'
      '%d'   = output col as ASCII decimal digits
      'H'    = output char 'H'
```

Overview of VLIB

VLIB (Video LIBrary) is the ZCPR3 library that provides the ZCPR3 programmer with a series of low-level routines for Z3TCAP access. VLIB is described in much more detail in the VLIB.HLP file, and this overview serves only to summarize its capabilities.

The VLIB routine Z3VINIT initializes VLIB for use with a ZCPR3 system. The address of the ZCPR3 environment descriptor is passed to the Z3VINIT routine in HL, and from that time onward all VLIB routines know the address of the Z3TCAP entry.

Some low-level functions provided by VLIB are:

Routine	*Function*
TINIT	Initialize terminal
DINIT	Deinitialize terminal
CLS	Clear screen
EREOL	Erase to End of Line
GOTOXY	Position Cursor
STNDOUT	Begin highlighting
STNDEND	End highlighting

Standard ZCPR3 TCAP File

The file Z3TCAP contains information on over 40 terminals. It is provided as a part of the ZCPR3 System, and it is used by TCSELECT. TCSELECT can display the names of the terminals contained in Z3TCAP and allow the user to select one, generating a *.Z3T file or storing the selection directly into memory for immediate use by the ZCPR3 System utilities.

TCAP Check Program

TCCHECK is used to check the Z3TCAP file for consistency. Its sole function is to ensure the validity of the Z3TCAP file and provide some statistics on it. A sample run of TCCHECK is shown below.

```
B4:SCR2>tccheck //
TCCHECK, Version 1.0TCCHECK - Select Entry from Z3TCAP.Z3T
Syntax:
        TCCHECK infile  -or-  TCCHECK infile.typ

where "infile" is the file to be checked by
the execution of TCCHECK. If no file type is
given, a file type of Z3T is the default.

Syntax:
        TCCHECK
where this alternate form may be used to check
```

```
the Z3TCAP.TCP file.

B4:SCR2>tccheck
TCCHECK, Version 1.0 File Z3TCAP    .TCP Not Found - Aborting
     --  Note: Z3TCAP.TCP MUST be in the same directory

B4:SCR2>tccheck root:z3tcap.tcp
TCCHECK, Version 1.0 File Z3TCAP    .TCP Not Found - Aborting
     --  Note: TCCHECK does not recognize named dirs

B4:SCR2>root:
A15:ROOT>tccheck
TCCHECK, Version 1.0
Z3TCAP File Check of Z3TCAP    .TCP Version 1.1
        File Checks with    44 Terminals Defined
```

TCAP Entry Definition Program

TCMAKE is used to create a *.Z3T file. Once created, the ZCPR3 utility LDR can load it into memory at the proper location (command is "LDR filename.Z3T"). A sample run of TCMAKE is shown below.

```
B4:SCR2>tcmake //
TCMAKE, Version 1.0
TCMAKE - Create a Z3T File
Syntax:
        TCMAKE outfile  -or-  TCMAKE outfile.typ

where "outfile" is the file to be generated by
the execution of TCMAKE. If no file type is
given, a file type of Z3T is the default.

B4:SCR2>tcmake myterm2
TCMAKE, Version 1.0

        ** Z3TCAP Main Menu for File MYTERM2 .Z3T **

Define: 1. Clear Screen Sequence
        2. Cursor Motion Sequence
        3. Clear to End of Line Sequence
        4. Standout Mode Sequences
        5. Terminal Init/Deinit Sequences
        6. Arrow Keys
        7. Terminal Name

Status: S. Print Status (Definitions so far)

Exit:   X. Exit and Write File
        Q. Quit and Abort Program without Writing File

Command? 2

Cursor Motion Definition
 1. Timing Delay
 Enter Delay Time in Milliseconds: 5
 2. Enter R if Row/Column or C for Column/Row: R
 3. Enter Equation for Row:    %+
 4. Enter Equation for Column: %+
 5. Enter Prefix Byte Sequence
  Char #1 - Type Char, .=Number, or <CR>=Done: Enter Number: 1B
  Char #2 - Type Char, .=Number, or <CR>=Done: Char =
  Char #3 - Type Char, .=Number, or <CR>=Done:
 6. Enter Middle Byte Sequence
  Char #1 - Type Char, .=Number, or <CR>=Done:
 7. Enter Suffix Byte Sequence
  Char #1 - Type Char, .=Number, or <CR>=Done:

        ** Z3TCAP Main Menu for File MYTERM2 .Z3T **

Define: 1. Clear Screen Sequence
        2. Cursor Motion Sequence
        3. Clear to End of Line Sequence
        4. Standout Mode Sequences
        5. Terminal Init/Deinit Sequences
        6. Arrow Keys
        7. Terminal Name

Status: S. Print Status (Definitions so far)

Exit:   X. Exit and Write File
        Q. Quit and Abort Program without Writing File
```

```
Command? 6

Arrow Key Definition
 Your Terminal's Arrow Keys may be defined ONLY
 if they generate only one character each. If they
 do, type Y to continue. If not, type anything else.
        Define Arrow Keys (Y/N)? Y
 Strike the Appropriate Arrow Key
 1. Arrow UP? ^K
 2. Arrow DOWN? ^V
 3. Arrow RIGHT? ^L
 4. Arrow LEFT? ^H

        ** Z3TCAP Main Menu for File MYTERM2 .Z3T **

Define: 1. Clear Screen Sequence
        2. Cursor Motion Sequence
        3. Clear to End of Line Sequence
        4. Standout Mode Sequences
        5. Terminal Init/Deinit Sequences
        6. Arrow Keys
        7. Terminal Name

Status: S. Print Status (Definitions so far)

Exit:   X. Exit and Write File
        Q. Quit and Abort Program without Writing File

Command? S

        ** Z3TCAP Status for File MYTERM2 .Z3T **

Review: 1. Clear Screen Definition
        2. Cursor Motion Definition
        3. Clear to End of Line Definition
        4. Standout Mode Definition
        5. Terminal Init/Deinit Definition
        6. Arrow Key Definition
        7. Terminal Name Definition
```

```
Exit:   X. Exit to Main Menu

Command? 1
Review of Clear Screen Definition
 1. Timing Delay = 0 Milliseconds
 2. Clear Screen Sequence:
  (1) ^[  1BH    (2) * 2AH
        Strike Any Key to Continue -

         ** Z3TCAP Status for File MYTERM2 .Z3T **

Review: 1. Clear Screen Definition
        2. Cursor Motion Definition
        3. Clear to End of Line Definition
        4. Standout Mode Definition
        5. Terminal Init/Deinit Definition
        6. Arrow Key Definition
        7. Terminal Name Definition

Exit:   X. Exit to Main Menu

Command? 2

Review of Cursor Motion Data
 1. Timing Delay = 5 Milliseconds
 2. Row or Column First: R
 3. Row Equation: -->%+ <--
 4. Column Equation: -->%+ <--
 5. Prefix Byte Sequence:
  (1) ^[   1BH  (2) = 3DH
 6. Middle Byte Sequence:
  -- Empty --
 7. Suffix Byte Sequence:
  -- Empty --
        Strike Any Key to Continue -

        ** Z3TCAP Status for File MYTERM2 .Z3T **

Review: 1. Clear Screen Definition
```

```
        2. Cursor Motion Definition
        3. Clear to End of Line Definition
        4. Standout Mode Definition
        5. Terminal Init/Deinit Definition
        6. Arrow Key Definition
        7. Terminal Name Definition

Exit:   X. Exit to Main Menu

Command? 6

Review of Arrow Key Definitions
 1. Arrow UP = ^K
 2. Arrow DOWN = ^V
 3. Arrow RIGHT = ^L
 4. Arrow LEFT = ^H
        Strike Any Key to Continue -

        ** Z3TCAP Status for File MYTERM2 .Z3T **

Review: 1. Clear Screen Definition
        2. Cursor Motion Definition
        3. Clear to End of Line Definition
        4. Standout Mode Definition
        5. Terminal Init/Deinit Definition
        6. Arrow Key Definition
        7. Terminal Name Definition

Exit:   X. Exit to Main Menu

Command? X

        ** Z3TCAP Main Menu for File MYTERM2 .Z3T **

Define: 1. Clear Screen Sequence
        2. Cursor Motion Sequence
        3. Clear to End of Line Sequence
        4. Standout Mode Sequences
        5. Terminal Init/Deinit Sequences
        6. Arrow Keys
        7. Terminal Name

Status: S. Print Status (Definitions so far)

Exit:   X. Exit and Write File
        Q. Quit and Abort Program without Writing File

Command? X
  Selected Terminal is: Rick's Terminal  -- Confirm (Y/N)? Y
File MYTERM2 .Z3T Created
```

TCAP Entry Selection Program

TCSELECT selects a terminal from the standard Z3TCAP file. The selected terminal may be loaded directly into memory or a *.Z3T file may be created. If a *.Z3T file is created, the ZCPR3 utility LDR can load it into memory at the proper location (command is "LDR filename.Z3T"). A sample run of TCSELECT follows.

```
B4:SCR2>tcselect //
TCSELECT, Version 1.0
TCSELECT - Select Entry from Z3TCAP.TCP

        TCSELECT outfile  -or-  TCSELECT outfile.typ

where "outfile" is the file to be generated by
the execution of TCSELECT. If no file type is
given, a file type of Z3T is the default.

Syntax:
        TCSELECT

where this alternate form may be used to store
the Z3TCAP entry for the selected terminal directly
into the Z3 Environment Descriptor.

Example 1: Create MYTERM.TCP

B4:SCR2>tcselect myterm
TCSELECT, Version 1.0

** Terminal Menu 1 for Z3TCAP Version 1.1 **
```

```
A.  AA Ambassador              K.  Concept 100
B.  ADDS Consul 980            L.  Concept 108
C.  ADDS Regent 20             M.  CT82
D.  ADDS Viewpoint             N.  DEC VT52
E.  ADM 2                      O.  DEC VT100
F.  ADM 31                     P.  Dialogue 80
G.  ADM 3A                     Q.  Direct 800/A
H.  ADM 42                     R.  General Trm 100A
I.  Bantam 550                 S.  Hazeltine 1420
J.  CDC 456                    T.  Hazeltine 1500

Enter Selection, + for Next, or ^C to Exit - +

** Terminal Menu 2 for Z3TCAP Version 1.1 **

A.  Hazeltine 1510             K.  P Elmer 1200
B.  Hazeltine 1520             L.  SOROC 120
C.  H19 (ANSI Mode)            M.  Super Bee
D.  H19 (Heath Mode)           N.  TAB 132
E.  HP 2621                    O.  Teleray 1061
F.  IBM 3101                   P.  Teleray 3800
G.  Micro Bee                  Q.  TTY 4424
H.  Microterm ACT IV           R.  TVI 912
I.  Microterm ACT V            S.  TVI 920
J.  P Elmer 1100               T.  TVI 950

Enter Selection, - for Last, + for Next, or ^C to Exit - +

** Terminal Menu 3 for Z3TCAP Version 1.1  **

A.  VC 404
B.  VC 415
C.  Visual 200
D.  WYSE 50

Enter Selection, - for Last, or ^C to Exit - -

** Terminal Menu 2 for Z3TCAP Version 1.1  **

A.  Hazeltine 1510             K.  P Elmer 1200
```

```
B.  Hazeltine 1520          L.  SOROC 120
C.  H19 (ANSI Mode)         M.  Super Bee
D.  H19 (Heath Mode)        N.  TAB 132
E.  HP 2621                 O.  Teleray 1061
F.  IBM 3101                P.  Teleray 3800
G.  Micro Bee               Q.  TTY 4424
H.  Microterm ACT IV        R.  TVI 912
I.  Microterm ACT V         S.  TVI 920
J.  P Elmer 1100            T.  TVI 950

Enter Selection, - for Last, + for Next, or ^C to Exit - T

  Selected Terminal is: TVI 950             -- Confirm (Y/N)? N

** Terminal Menu 2 for Z3TCAP Version 1.1  **

A.  Hazeltine 1510          K.  P Elmer 1200
B.  Hazeltine 1520          L.  SOROC 120
C.  H19 (ANSI Mode)         M.  Super Bee
D.  H19 (Heath Mode)        N.  TAB 132
E.  HP 2621                 O.  Teleray 1061
F.  IBM 3101                P.  Teleray 3800
G.  Micro Bee               Q.  TTY 4424
H.  Microterm ACT IV        R.  TVI 912
I.  Microterm ACT V         S.  TVI 920
J.  P Elmer 1100            T.  TVI 950

Enter Selection, - for Last, + for Next, or ^C to Exit - S

  Selected Terminal is: TVI 920             -- Confirm (Y/N)? Y

File MYTERM  .Z3T Created

     -- Example 2: Select terminal and store it in memory
B4:SCR2>tcselect
TCSELECT, Version 1.0

** Terminal Menu 1 for Z3TCAP Version 1.1  **

A.  AA Ambassador           K.  Concept 100
```

```
B.  ADDS Consul 980           L.  Concept 108
C.  ADDS Regent 20            M.  CT82
D.  ADDS Viewpoint            N.  DEC VT52
E.  ADM 2                     O.  DEC VT100
F.  ADM 31                    P.  Dialogue 80
G.  ADM 3A                    Q.  Direct 800/A
H.  ADM 42                    R.  General Trm 100A
I.  Bantam 550                S.  Hazeltine 1420
J.  CDC 456                   T.  Hazeltine 1500

Enter Selection, + for Next, or ^C to Exit - +

** Terminal Menu 2 for Z3TCAP Version 1.1  **

A.  Hazeltine 1510            K.  P Elmer 1200
B.  Hazeltine 1520            L.  SOROC 120
C.  H19 (ANSI Mode)           M.  Super Bee
D.  H19 (Heath Mode)          N.  TAB 132
E.  HP 2621                   O.  Teleray 1061
F.  IBM 3101                  P.  Teleray 3800
G.  Micro Bee                 Q.  TTY 4424
H.  Microterm ACT IV          R.  TVI 912
I.  Microterm ACT V           S.  TVI 920
J.  P Elmer 1100              T.  TVI 950

Enter Selection, - for Last, + for Next, or ^C to Exit - T

  Selected Terminal is: TVI 950          -- Confirm (Y/N)? Y

 ZCPR3 Environment Descriptor Loaded
```

Appendix A

Glossary of Terms

Absolute Path
A path which is expressed in exact terms with no relative disk or user area references. A path like "A1 A15" is an absolute path. See RELATIVE PATH.

Ada
The standard programming language for Embedded Computer Systems in the US Department of Defense. Pascal is Ada's primary ancestor, but Ada has many extensions over conventional Pascal.

Alias
A program which expands into a sequence of commands with parameter substitution from the original command line. Such a program is created by the ALIAS Tool.

BDOS
The Basic Disk Operating System part of CP/M or ZCPR3. This is the part of the operating system which manages the files and provides some higher-level input/output functions.

BIOS
The Basic Input/Output System part of CP/M or ZCPR3. This is the part of the operating system which manages the devices and provides a low-level, machine-independent interface to the disks and character-oriented input/output devices.

Boot, Cold
The process of initializing the operating system when the computer is first turned on. The BIOS contains a cold boot routine which is used to initialize many features of the system, including several of the ZCPR3 buffers.

Boot, Warm
The process of reinitializing the operating system at some time after it has begun execution. The BIOS contains a warm boot routine, but this routine does not usually have any effect on the ZCPR3 environment except for reloading the ZCPR3 Command Processor from disk.

CBIOS
Customized Basic Input/Output System. This is a term used to represent the BIOS which has been adapted for a particular microcomputer. See BIOS.

COM File
An executable binary image which executes at memory location 100H and is loaded from disk by the ZCPR3 Command Processor. Once loaded, a COM file is called by the ZCPR3 Command Processor like a subroutine.

Command
A statement issued by the user or a program which results in a program to be executed. A command consists of a verb followed by zero or more arguments. The verb uniquely identifies the name of the program to execute.

Command Package
A collection of routines which are stored in one logical binary entity. Each routine acts like a COM file, and a table at the beginning of the package identifies the name of each routine and its starting address.

Command Search Hierarchy
See HIERARCHY.

Device Management
The control of the physical devices associated with a computer system. Device management concerns itself with the control of disk drives and input/output devices (such as computer terminals).

Directory
A logical entity in which files are stored. A directory is identified by a disk letter and a user area number. A physical directory is sometimes also reference, and this is the location on disk where information about the files on that disk is stored.

DIR Form
A mnemonic used to refer to a disk and user area. ROOT: is a DIR form.

DU Form
A reference to a directory in terms of a disk and user area number combination. A15: is a DU form.

Error Handler
A program which is invoked by the ZCPR3 Command Processor when an error is encountered in the current command. A pointer to the current command is provided for the Error Handler to use.

Flow Command Package
A collection of routines which implement commands that affect the IF state of the ZCPR3 environment. The Flow Command State is set to TRUE, FALSE, or null (no active IF) by these commands. Commands such as IF and ELSE are implemented in Flow Command Packages. See COMMAND PACKAGE.

Flow Command State
See FLOW STATE.

Flow State
The state of command execution under ZCPR3. If the Flow State is TRUE, any command can execute. If the Flow State is FALSE, only Shells and Flow Command Package routines can execute.

Hierarchy
A structured sequence of objects. This term is usually applied to the Command Search Hierarchy, which is the sequence of steps taken in order to process a command under the ZCPR3 System.

Information Management
The control of the resources of a computer which deal with the data contained in the system. Specifically, information management deals with the control of files and directories.

Input/Output Package
A logical grouping of routines which control a set of logical devices that are implemented as combinations of one or more physical devices. An I/O Package is called by the BIOS to provide access to character-oriented devices, such as terminals.

Memory Image
A representation of a program as it resides in the memory of the computer. This is as opposed to a SYSGEN image, which is the form the operating system takes as it resides on disk, or the disk-based image, which is the form a program takes as it resides on disk.

Memory Management
The control of the allocation of memory inside a computer system. The CP/M and ZCPR3 systems perform very little in the way of memory management aside from outlining a recommended partitioning of the memory.

Message
A unit of information which is stored in one or more byte in a pre-defined location in memory. Programs can pass information between each other and the ZCPR3 Command Processor by means of messages under the ZCPR3 System.

Multiple Command Line
A sequence of one or more commands, where each command is separated from another by a semicolon. "XDIR;DIR" is a multiple command line.

Named Directory
A disk and user area (directory) that has a mnemonic associated with it. ROOT is the DIR form which is associated with the named directory at disk A, user area 15 by convention.

Operating System
A program which manages the four basic resources of a computer system: devices, information, memory, and processes and processors. CP/M and UNIX are operating systems.

Package
A logical collection of data and routines that is divided into two parts: a visible section and a hidden section. The interface to the package routines is defined in the visible section and the routines themselves are in the hidden section. The concept of a package was taken from the programming language Ada.

Parsing
The process of breaking up a command into its representative tokens and storing these tokens into the appropriate buffers. A command consists of a verb, up to two file references, and a series of options in most cases. Each of these items is a token that is extracted by the parser in the ZCPR3 Command Processor.

Path
A sequence of directory references which is examined when searching for a COM file in for certain other files in various situations. See ABSOLUTE PATH and RELATIVE PATH.

Process
A program in execution. A program is a passive entity, which is simply an expression of a sequence of commands. A process is an active entity which can cause physical activity to occur, such as devices engage. See PROGRAM.

Process Management
The control of the processes inside a computer system. The CP/M and ZCPR3 operating systems do very little in the way of process management aside from loading a program and causing it to execute. See PROCESS.

Program
An expression of a sequence of commands that may be translated into a form that can be executed on a computer. See PROCESS.

Relative Path
A sequence of directory references that includes at least one entry whose disk or user area reference is represented by a "$" character, which refers to the disk or user area which is currently logged into. The resolution of a relative path depends upon the location of the user when he issues a command, and the absolute expression of this path is based upon the location of the current directory.

Resident Command Package
A collection of routines which are located in memory and may be invoked by the ZCPR3 Command Processor as needed. See PACKAGE and FLOW COMMAND PACKAGE.

Script
A textual expression that is translated into a series of one or more commands. An Alias generates a script.

Shell
A front end program which accepts commands from the user and preprocesses these commands before passing them on to the ZCPR3 Command Processor for resolution and execution. VFILER and SH are two different Shells under the ZCPR3 System.

Symbolic Path
See RELATIVE PATH.

Tool
A program which is used to perform a useful function in a general-purpose sense. All of the ZCPR3 programs are tools, as opposed to games.

Tool Set
A collection of tools, where the tools are usually related in functionality or operation to each other.

Virtual Machine
A machine which exists at a higher level of abstraction than the actual physical machine which is supporting it. The virtual machine concept is used to raise the details of interfacing with a machine to a higher level of abstraction so that the intimate details of a machine need not be of concern.

Appendix C

References

The following sections outline other sources of information on ZCPR2, ZCPR3, SYSLIB2, and SYSLIB3 that may be useful to the installers and users of ZCPR3.

ZCPR2 and SYSLIB2 Publications and Documentation

ZCPR2 Manuals

Conn, Richard. 'Concepts Manual for ZCPR2 – Z80 Command Processor Replacement, Version 2,' *Manual Revision 0*, 3 February 1983, 21 pages.

Conn, Richard. 'Installation Instructions for ZCPR2 – Z80 Command Processor Replacement, Version 2,' *Manual Revision 0*, 2 February 1983, 48 pages.

Conn, Richard. 'Rationale Manual for ZCPR2 – Z80 Command Processor Replacement, Version 2,' *Manual Revision 0*, 26 February 1983, 65 pages.

Conn, Richard. 'User's Guide for ZCPR2 – Z80 Command Processor Replacement, Version 2,' *Manual Revision 0*, 4 February 1983, 138 pages.

SYSLIB2 Manuals

Conn, Richard. *SYSLIB User and Reference Manual for SYSLIB Version 2.4*, 4 February 1983, 112 pages.

Conn, Richard. *User's Guide to SYSLIB 2.3*, Revision B, 14 December 1982, 56 pages.

Software Upgrades to SYSLIB2 and ZCPR2

Four upgrades to SYSLIB2 and ZCPR2 were issued. There were all written by Richard Conn, and their dates are:

4 March 1983
30 March 1983
30 April 1983
22 June 1983

Sources

Contact the following sources for copies (on disk or hard copy, depending on the source) of the above-mentioned documentation.

SIG/M (disks)
New York Amateur Computer Club (hardcopy)
SIMTEL20 (DDN access)

CP/M Books

The following books on CP/M are recommended.

Hogan, Thom. *Osborne CP/M User Guide*, Osborne/McGraw-Hill, 1981, 283 pages.
Johnson-Laird, Andy. *The Programmer's CP/M Handbook*, Osborne/McGraw-Hill, 1983, 501 pages.

ZCPR3 Sources

The following are the addresses of sources for information on ZCPR2, ZCPR3, SYSLIB2, and SYSLIB3.

Selected Computer Clubs

SIG/M provides a library of software on a number of formats of floppy disk. They are a source for the ZCPR2, SYSLIB2, ZCPR3, and SYSLIB3 software.

The New York Amateur Computer Club has published hardcopy of the ZCPR2 and SYSLIB2 documentation.

ACGNJ and SIG/M

SIG/M-Amateur Computer Group of New Jersey
PO Box 97
Iselin, NJ 08830

New York Amateur Computer Club

New York Amateur Computer Club, Inc
PO Box 106
New York, NY 10008

Echelon, Inc.

Echelon, Inc
101 First Street
Los Altos, CA 94022
(415) 948-5321

Echelon, Inc has been selected as exclusive agent for licensing commercial uses of ZCPR3. Companies who wish to incorporate ZCPR3 into their products should contact Echelon for licensing arrangements.

Echelon also distributes ZCPR3 (including SYSLIB3) for non-commercial use. Individual users may obtain copies of the programs for essentially cost of disks plus handling and mailing expenses.

Echelon will soon provide a computerized bulletin board service in support of ZCPR3. Information on how to acquire a copy of the system, upgrades and changes to the system, and general user/creator feedback and communication will be supported by this bulletin board.

At irregular intervals Echelon distributes a ZCPR3 Newsletter to customers who have purchased ZCPR3 or SYSLIB. It provides news of updates, bugs and their fixes, and hints and tips from users.

New York Zoetrope, Inc.

New York Zoetrope, Inc.
Suite 516
80 East 11th St.
New York, NY 10003
(212) 420-0590

Cable: NYZOETROPE, N.Y.
Source: TCN 121

New York Zoetrope, Inc, is the publisher of the hardcopy documentation on ZCPR3 and SYSLIB3. Contact New York Zoetrope for details.

Magazine Articles on ZCPR3

If you are interested in finding out more about ZCPR3, the following references are cited. This list is not complete, but does provide some pointers to reviews and other information.

Blum, Robert. 'CP/M Exchange,' *Dr. Dobb's Journal*, Number 81, July 83. Robert is doing a continuing series on SYSLIB3 and other libraries associated with ZCPR3.

Bove, Tony and Rhodes, Cheryl. 'What is ZCPR3?,' *User's Guide magazine, Issue 11/12, Vol 2.5/2.6, Dec 84.*

Conn, Richard. 'The Evolution of ZCPR, Part 1,' Computer Language magazine, Vol 1, Number 2, Oct 84.

Conn, Richard. 'The Evolution of ZCPR, Part 2,' *Computer Language magazine*, Vol 1, Number 3, Nov 84.

Gerrold, David. 'Up and Running,' *Profiles magazine*, Nov 84. David is doing a continuing series on ZCPR3.

Haigwood, Jerry. 'The Ampro Bookshelf Computer,' *User's Guide magazine*, Issue 9, Vol 2.3, July 84.

Wright, Dennis. 'ZCPR3,' *Computer Language magazine*, Vol 1, Number 3, Nov 84.

ZCPR3 Newsletters

A very nice source of information is the ZCPR3 Newsletters of Echelon. They come out every 2 weeks (more or less), and they are released to the public. Many RCP/M systems carry them, and Echelon provides them to customers in hardcopy form.

ZCPR3 Configuration Management

ZCPR3 is being placed under configuration management, and the CM system is about ready. Reports on latest versions of all the software, CRCs, etc, will be generated by this. Contact Echelon for details.

ZCPR3 Electronic Bulletin Board

The number for the ZCPR3 bulletin board associated with Echelon is: 415-489-9005

There are a growing number of ZCPR3 bulletin boards sprouting up. Watch for them and their screen-oriented displays.

Listed here are a few of the bulletin boards devoted to ZCPR3.

You'll find useful information and help here.

Jay Sage	Boston, MA	617/965-7259
Jud Newell	Toronto, CANADA	416/232-0442
Richard Rodeheaver	Reynoldsburg, OH	614/864-2673
Robert Tate	Orlando, FL	305/831-6049
Richard Petersen	El Paso, TX	915/755-3342
Al Hawley	Los Angeles, CA	213/670-9465
Richard Mead	Pasadena, CA	818/799-1632
David McCord	Fremont, CA	415/489-9005
Tim Linehan	Olympia, WA	206/357-7400

Appendix D

Obtaining Free Software

MAINFRAME PROGRAMS FOR TRANSFERRING FILES BETWEEN MAINFRAMES AND MICROS

Christensen Protocol:

For a micro to reliably exchange files with a mainframe, cooperating file transfer programs with automatic error detection and retransmission of faulty blocks must be running on both computers. One such family of programs uses a popular protocol created by Ward Christensen and enhanced by others. Two popular programs, UC and the older UMODEM (both written in C), implement this protocol on UNIX machines.

On ITS machines, file transfer using the Christensen protocol can be done using MMODEM (type :MMODEM for instructions), or LMODEM. Documentation for LMODEM is in file which types an ASCII file stored in ITS binary format; TYPESQ, which types an ITS binary format "squeezed" file (see the first paragraph under FILE TYPES); USQ, which creates an unsqueezed version of a squeezed file; HEXIFY, which creates an Intel hex format file from an ITS binary format COM file; COMIFY which creates a COM file from an Intel hex file; and CRC, which computes the Cyclic Redundancy Check value for a file, using the same algorithm that is used by the CP/M program CRCK. Brief instructions for any of these utilities except LMODEM can be obtained by typing ":utility_name" (for example, :CRC).

Several other utilities for transferring and manipulating files are commonly available for TOPS-20 and VAX/VMS systems. Your local system support people should be able to tell you about the available programs.

Kermit:

Another excellent program for transferring files is called KERMIT. This program has the advantage of being available for an impressively large number of mainframes and micros. It is, for example, available for the IBM-PC, and it DOES NOT require CP/M.

MICROCOMPUTER PROGRAMS FOR TRANSFERRING FILES BETWEEN ' MAINFRAMES AND MICROS

Christensen Protocol:

An excellent program for transferring files between micros, or between micros and mainframes is called MODM7xx, where the "xx" is replaced with two digits to give the current version number. This program, often referred to as MODEM7 (the name of its easier to pronounce ancestor), uses the popular Christensen protocol to transfer files with automatic error detection and retransmission of erroneous blocks.

And then, there is MEX. MEX stands for "modem executive", and it is just what the name implies, a communications and file-transfer program with a built-in mini operating system that runs under CP/M. This program can do file transfers using either the Christensen or Compuserve protocol, and it has an enormous potential for highly automated operations because it can read and execute command-scripts pre-stored in disk files. These scripts can include sending commands to a remote computer, as if they had been sent manually from the microcomputer in terminal-mode. Users of this relatively new program are still exploring its possibilities.

Kermit:

As stated earlier, KERMIT is also an excellent program for transferring files between computers. It, too, does automatic error detection and retransmission, and it works between mainframes and micros, between micros, and between mainframes. See the earlier "Kermit" paragraph for details.

Getting Started:

In order to get MODM7xx, MEX or KERMIT running on your micro, you must first transfer the necessary files from mainframe to micro. If you already have a receive-to-disk communications program of some sort, you can use it to move the needed files. It is VERY CONVENIENT to be able to transfer 8-bit binary files, although in most cases it is not absolutely necessary. Some of the files are quite large. For example, MODM7xx.COM is over 18K bytes, and the HEX file (which you will need if you can't transfer 8-bit files) is over 45K. Moving large files to your micro without using an error detecting protocol can result in frustrating errors, but it can be done by receiving multiple copies and using manual or machine-assisted comparisons to locate and repair bad parts of the code.

MAGAZINE ARTICLES

The following magazine articles are highly recommended for those who have never used an RCP/M system before to download or upload files.

Tony Bove, Cheryl Rhodes, and Kelly Smith. 'Calling Bulletin Boards for Free Software', *User's Guide magazine*, Issues 11/12, Vol 2.5/2.6, December 1984.

Tony Bove, Cheryl Rhodes, and Kelly Smith. 'Using Modem7 (MDM720)', *User's Guide magazine*, Issues 11/12, Vol 2.5/2.6, December 1984.

Benjamin Cohen. 'Free Software, The Newest Modem 7 (MDM740)', *User's Guide magazine*, Issues 11/12, Vol 2.5/2.6, December 1984.

Kim Levitt. 'Phone List of RCP/M Bulletin Board Systems, Revision 50', *User's Guide magazine*, Issues 11/12, Vol 2.5/2.6, December 1984.

'Free Software: Getting It', *User's Guide magazine*, Issues 11/12, Vol 2.5/2.6, December 1984.

Downloading the Easy Way

by David L. Mix

In your manual look for the way to establish a connection between your computer and the RCP/M, and the section on "XMODEM" or "MODEM" Protocol. This protocol gives you automatic high-level error-checking to assure that you get files that are not affected by telephone line noise. Almost all boards work with 300 baud, and many now work with 1200 baud. If you aren't sure, try 300 baud, or just use the default. Usually the other defaults work with Bulletin Boards, so for now skip the other technical matters. Expect to supply your Name, Address, and a password you can remember. From page 338 of this book, get the phone numbers of some local boards. In my training sessions with beginners, we have a saying, "Read the Fine Screen" or, "RFS". If you get stuck, the information is usually on the screen. This is important, since different boards offer a rich variety of ways to do the same thing. Once you get started, it adds to your learning. Most boards have some kind of HELP. You can see many of the commands you will need by using DIR on the A drive. Files for you to transfer are on the B or higher drives.

Most boards need you to hit the RETURN key twice either before or after they identify themselves. If you are asked "How many nulls do you need?", answer with the number zero. A Welcome Message will tell you things you need to know to get to CP/M. After you get to CP/M on the board, build your confidence by changing to the B drive with 'B:'. Look over that User Area with DIR, or DIR $AD *.* for everything on all drives. Files that end in .OBJ have been renamed from .COM. You will not be able to use all commands like the command to erase, ERA, for obvious security reasons. You will be able to TYPE a screenfull or two of files ending in .DOC or Documentation Files for information about using related programs. To actually get a program, pick a short file, and issue this command to the Board: XMODEM S FILENAME. When the board is ready it will reply with a message giving you the file size and transfer time. If you are using MODEM7 software, use CTRL E, then R FILENAME, automatically changing to Xmodem Protocol on your end. With some software, you will have to switch to Xmodem Protocol on your end. You get off the board by typing BYE or G.

Congratulations, you just transferred you first file!

Z-SYSTEM OPERATING SYSTEM MODULES AVAILABLE FROM NEW YORK ZOETROPE

Developed by Echelon Inc.

ITEM		PRICE
1	ZCPR3 Core Starter Kit. Contains source to permit manual installation of ZCPR3 console command processor and its buffers using MOVCPM, DDT, MAC and SYSGEN. Loose-leaf 173-page Installation Guide included, with 20 utility programs in binary form	$39.00
2	ZCPR3 Utilities Package consists of 70 programs in both binary and source code form. Combined with Item #1 forms complete processing sub-system. Online (over 400K-byte) help system included	$89.00
3	Z3-Dot-Com is the auto-install version of ZCPR3. Package contains all binary utility programs released to date, complete system. Secure versions available. Installs in four minutes or less; full online help files, 52-page loose-leaf tutorial	$149.00
4	Z3-Dot-Com on one disk for those who already have Item #2 and wish to add auto-install to their collection	$49.95
5	Z-Com Same as Item #3 but with Item #9 added	$199.00
6	Z-Com Same as Item #4 but with Item #9 added	$99.00
7	Z-Com2 Same as Item #3 but with Item #10 added	$219.95
8	Z-Com2 Same as Item #4 but with Item #10 added	$119.95
9	ZRDOS Improved BDOS in binary, with six support utilities: copy file, backup and archiving, set and display file attributes; file compare, dump, and bi-directional viewing. Upward compatible with CP/M-80. 66K-bytes of online help for both functions and utilities, plus 35-page loose-leaf manual for programming system functions	$49.50
10	ZRDOS2 Same as Item #9 but with single level re-entrance, function 10 full command line editing and buffer recall, get DMA address, and 512-megabyte maximum file and disk sizes, 45-page loose-leaf manual	$74.99

1. Name ______________________ Date __________

   Street ______________________ Telephone __________

   City ______________ State __________ Zip ________

2. Z80 CP/M based computer (yes/no) Computer type/brand __________

3. Disk format: 8 inch SSSD IBM 3740 standard (yes/no)

   or

   5 1/4 inch, 40 tracks, soft-sectored, single-sided double-density: (yes/no)

Computer format:	☐ Kaypro	☐ Osborne1	☐ Epson QX-10	☐ Heath/Zenith 89/90/100
(check one)	☐ Ampro	☐ Morrow	☐ Apple CP/M	☐ Northstar CP/M (hard)

Any Special instructions? ______________________

4. Ordering:

Item Number & Name	Quantity	Unit Price	Extension
		Subtotal $	
NY State residents please add 8 1/4% sales tax			
	Total enclosed		

Send orders to:
NEW YORK ZOETROPE
Suite 516/80 East 11th Street
New York, NY 10003

Please allow 6 weeks for delivery. Thank you.
Credit card purchases: call (212) 420-0590

Index