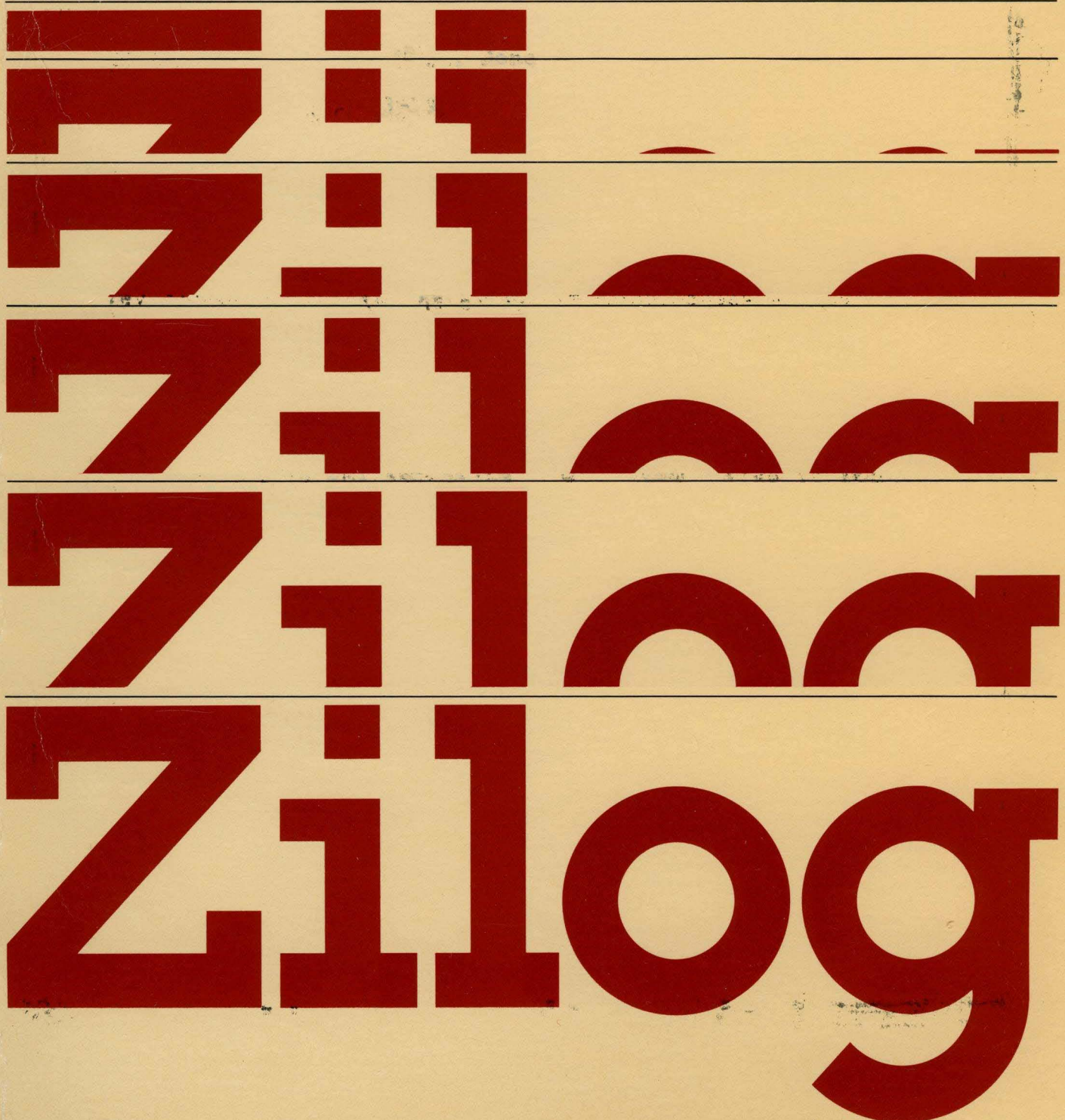
Z80-RIO

Operating System User's Manual

September 1978



03-0072-01
Revision A
September 1978

Z80-RIO

Operating System User's Manual

September 1978

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION AND OVERVIEW 1

1.1 INTRODUCTION 1

1.2 SYSTEM OVERVIEW 3

1.2.1 Hardware Configuration 3
1.2.2 File Systems 3
1.2.3 System Initialization 6
1.2.4 Commands 7
1.2.5 I/O 7

CHAPTER 2 - RIO EXECUTIVE 9

2.1 SYSTEM INITIALIZATION 9

2.2 FILE NAME CONVENTIONS 10

2.3 MEMORY MANAGEMENT 12

2.3.1 MEMMGR 13

2.4 COMMAND STRING INTERPRETATION 13

2.5 ERROR HANDLING 15

2.6 PROGRAM EXECUTION OF COMMANDS 15

CHAPTER 3 - I/O STRUCTURE 16

3.1 OVERVIEW 16

3.2 I/O REQUESTS - SYSTEM CALLS 17

3.3 THE 'ASSIGN' I/O REQUEST 19

3.4 STANDARD RIO I/O DEVICES 21

3.4.1 ZDOS 21
3.4.2 DFS 21
3.4.3 NULL 21
3.4.4 CON 22
3.4.5 PCON 27
3.4.6 FLOPPY 27
3.4.7 DISK 27

CHAPTER 4 - PROGRAM INTERFACE 28

4.1 PROGRAM LOCATION 28

4.2 PARAMETER STRING ADDRESS 29

4.3 PROGRAM STACK SPACE 29

4.4 PROGRAM TERMINATION - ERROR HANDLING . . . 29

4.5 SYSTEM CALLS - SYSTEM ENTRY POINT 30

4.6 INTERRUPT STATUS 31

4.7 I/O UNIT UTILIZATION 31

4.8 PROGRAM EXAMPLES 32

CHAPTER 5 - RIO COMMANDS 33

5.1 ACTIVATE 35

5.2 ALLOCATE 37

5.3 BRIEF 39

5.4 CAT 40

5.5 CLOSE 44

5.6 COMPARE 45

5.7 COPY 47

5.8 COPY.DISK 49

5.9 COPYSD 51
5.10 DATE 53
5.11 DEACTIVATE 54
5.12 DEALLOCATE 55
5.13 DEBUG 56
5.14 DEFINE 60
5.15 DELETE 63
5.16 DISK.FORMAT 67
5.17 DISK.REPAIR 70
5.18 DISK.STATUS 72
5.19 DISPLAY 74
5.20 DO 75
5.21 DUMP 79
5.22 ECHO 80
5.23 ERROR 81
5.24 ERRORS 82
5.25 EXTRACT 83
5.26 FORCE 84
5.27 FORMAT 85
5.28 HELP 88
5.29 IMAGE 89
5.30 INITIALIZE 90
5.31 LADT 91

5.32 MASTER 92
5.33 MOVE 93
5.34 PAUSE 98
5.35 RELEASE 99
5.36 RENAME 100
5.37 RESTORE_TABS 102
5.38 SAVE_TABS 103
5.39 SET 104
5.40 STATUS 109
5.41 VERBOSE 111
5.42 XEQ 112
5.43 EXPRESSION EVALUATION 113

CHAPTER 6 - ZDOS 114
6.0 ZDOS OPERATION 114
6.1 INITIALIZE 122
6.2 ASSIGN 123
6.3 OPEN 125
6.4 CLOSE 132
6.5 REWIND 134
6.6 READ BINARY 135
6.7 WRITE BINARY 137
6.8 WRITE CURRENT 139
6.9 DELETE 140

6.10 DELETE REMAINING RECORDS 142

6.11 ERASE 143

6.12 READ AND DELETE 145

6.13 READ CURRENT 147

6.14 READ PREVIOUS 149

6.15 READ DIRECT 151

6.16 SKIP FORWARD 153

6.17 SKIP BACKWARD 155

6.18 SKIP TO END 157

6.19 RENAME 158

6.20 UPDATE 160

6.21 SET ATTRIBUTES 162

6.22 QUERY ATTRIBUTES 164

CHAPTER 7 - DFS 166

7.1 ZILOG DISK CONTROLLER 166

7.2 DFS OPERATION 168

7.3 SOFTWARE ORGANIZATION 171

7.4 DFS ALLOCATION 172

7.4.1 Sector 0 Format 172
7.4.2 DFS Allocation Algorithm 173

7.5 THE BARE DISK CONTROLLER 174

7.6 CONTROLLER BOOTSTRAP OPERATION 177

7.7 SYSTEM BOOTSTRAPPING on the MCZ-1/35 . . 179

APPENDICES

APPENDIX A - RIO/ZDOS/DFS Error Codes
APPENDIX B - RIO Command Syntax Summary
APPENDIX C - RIO System Constants
APPENDIX D - Converting Files to RIO Format
APPENDIX E - Altering Default RIO
APPENDIX F - I/O Request Vector Format and
I/O Request Codes
APPENDIX G - Program Examples
APPENDIX H - Internal Command Table Contents
APPENDIX I - RIO Memory Manager
APPENDIX J - Descriptor Record of Procedure Type File
APPENDIX K - ZDOS/DFS Command Summary

PREFACE

This manual provides an introduction and user's manual for the RIO operating system used with Zilog's Micro (ZDS). Detailed description is provided for system features, including the bootstrap process, the RIO Executive, default console drivers, I/O structure, program interface, and the Zilog Floppy Disk File System, ZDOS, and the Zilog Hard Disk File System, DFS.

Other pertinent documentation with which the reader may want to become familiar includes:

Z80-MCZ PROM User's Manual

Z80-ZDS PROM User's Manual

Z80-RIO Relocating Assembler and Linker User's Manual

Z80-RIO Text Editor User's Manual

This manual makes use of the following conventions of notation:

Optional portions of a modifier are enclosed in brackets, [].

The symbol for logical or, '|', is used if either option can be issued. STATUS [0 | 1...7] means the command can be issued as STATUS 0, STATUS 1,... STATUS 7, or simply as STATUS.

Parameters which can be repeated zero or more times are enclosed in parentheses and followed by an asterisk - e.g., (filename)*.

Parameters which can be repeated as necessary but must appear at least once are enclosed in parentheses and followed by a plus sign - e.g., (filename)+.

All memory addresses and constants referring to memory allocation are given in hexadecimal. Unless so specified, other constants are given in decimal. Hexadecimal constants are also indicated by an 'H' immediately following the hex digits, e.g., 4FH.

CHAPTER 1

INTRODUCTION AND OVERVIEW

1.1 INTRODUCTION

The Z80 Operating System with Relocatable Modules and I/O Management, or RIO, is a general-purpose computing system designed to facilitate the development and integration of user's programs into a production environment. RIO is available on various Zilog hardware configurations including the Z80 Micro Computer System (MCZ-1) series and the Z80 Development System (ZDS). The Z80 Development System provides extensive hardware debugging aids to assist the engineer/programmer in Z80-based hardware/software system design. The user has a choice between a modest environment with a minimum of system support or an enhanced environment which provides access to an assortment of system support utilities including the Zilog Floppy Disk File System, ZDOS, and the Zilog Hard Disk File System, DFS.

In the modest environment, the user has access to 3K (1K=1024) bytes of dedicated read-only memory which contains a program debugger with file manipulation capability, a floppy disk driver which supports up to eight disk drives, and a basic console driver with provision for paper tape operation.

In the enhanced environment, the user also has access to the RIO Executive, ZDOS, DFS, and a collection of disk-resident software including a text editor, macro assembler, and linker. The RIO Executive provides standardized I/O management permitting device independent program development and utilization of alternate or multiple file systems. ZDOS provides a versatile floppy disk based file system with variable record length files; up to 16 concurrently active files; management of user-defined scratch files which are automatically deallocated after use; and support of up to eight disk drives for over 2.5 megabytes of on-line storage. The hard disk file system,

DFS, supplies similar features on 10 megabyte high speed disks. The text editor, macro assembler and linker give the user full support in program development, minimizing assembly time with relocatable modules while allowing complex memory overlay structures. In addition, a console driver is provided which allows user definition of character delete and line delete symbols, automatic insertion of any number of line feeds, and automatic echo mode to accommodate a wide range of console devices.

1.2 SYSTEM OVERVIEW

1.2.1 HARDWARE CONFIGURATION

The RIO Operating System is designed to operate with the 4K PROM in either the Zilog Micro Computer System (MCZ) or Development System (ZDS). A minimum configuration of 32K (1K=1024) of random access memory, one disk drive, and a console device is required.

The MCZ 1/20 Zilog Micro Computer System is equipped with two floppy disk drives. The left drive is designated drive '2' and the right drive is designated drive '0'. The Zilog Development System (ZDS) also has two drives, but the designations are '1' and '0' for left and right, respectively.

The MCZ 1/35 uses hard disk cartridge and fixed platter drives. The usual configuration consists of one fixed platter and one cartridge drive, designated '0' and '1', respectively. The system will support up to 8 drives.

1.2.2 FILE SYSTEMS

For systems using floppy disks, ZDOS controls the organization and allocation of the sectors on a diskette. While the basic unit of disk allocation is the sector, the fundamental structure within ZDOS is the 'file'. A file consists of zero or more sectors of data which contain logically-related information. Each file has a set of attributes including a name of from one to thirty-two characters, a set of (possibly null) properties, a type, a subtype, and a record length. The smallest amount of information that can be read from or written to the disk is the contents of one sector, but more efficient operation can often be achieved by grouping from one to thirty-two contiguous sectors (a complete track) into one unit which is then read or written together. This unit is called a 'record' and the number of bytes of data in the record is the record length. A record may consist of 1, 2, 4, 8, 16 or 32 sectors; therefore the record length may be 128, 256, 512, 1024, 2048, or 4096 bytes.

On systems with hard disks, the Disk File System (DFS) provides a similar file structure. The bulk of the DFS

software runs in the memory of an intelligent disk controller; only a small interface routine resides in the main system memory, thus resulting in a large memory space saving. DFS files have the same structure as ZDOS files, except that multiple-sector records are not supported. The sector (and record) size is 512 bytes.

The properties of a file are defined by the user and may include any combination of the following:

1) write protected - may not have contents altered;

2) erase protected - may not have contents deleted;

3) locked attributes - may not have its attributes changed (a file's attributes include its properties, type, subtype, and other information included in the file's 'descriptor record'; see below);

4) random - file is in a format for random access;

5) secret - file is not normally found in directory searches (see below).

When a file is created, the user specifies its type, which must be exactly one of the following:

1) directory - a file directory (see below);

2) procedure - file contains information which can be loaded into memory and executed directly;

3) ASCII - file consists of symbols encoded in the American Standard Code for Information Interchange format, such as those produced by the editor or console input device;

4) binary - data of an unspecified format.

In addition to the file type, the user may define a subtype, which is a value ranging from 0 (default) to 15. The subtype is useful to differentiate between files of a certain type. For instance, RIO requires all I/O device files to be of type procedure, subtype 1.

The file system maintains a special file on each disk which is named 'DIRECTORY'. In this file are the names of all files (including itself) on the disk and the location of the first record of each file. The first record of each file is one sector long (regardless of the record length of the file) and is called the 'descriptor record'. All the file attributes including entry point (where execution may begin), date of creation, date of last modification, first data record address, last data record address, record length, and record count are contained in this record. Each record of the file contains pointers (disk addresses) to the previous record and the subsequent record in the file. Note that records which are logically in order according to file contents may, in fact, reside in an arbitrary order on the disk. This 'linked' structure allows maximum utilization of the disk. The disk allocation algorithm in ZDOS attempts to localize the disk sectors used for a single file. Note that the sectors which comprise a single file record are physically contiguous on the disk and are therefore always read or written as a single disk access.

ZDOS maintains a bit map to keep track of allocated vs. unallocated disk sectors. This map resides on three sectors of the diskette which are preallocated by the diskette formatting utility and is read into memory by the Initialize command or automatically by ZDOS when the diskettes are exchanged. The map is written from memory to the diskette when a file is closed following an allocation change. If the diskette is formatted as a 'system' disk, additional space is preallocated for the system bootstrap routine and the GET/SAVE command package (see the Debug command, section 5.13).

Under DFS, the unallocated hard disk sectors are linked in a free chain. The allocation and deallocation of sectors is a matter of removing sectors from or adding sectors to the free chain. System disks contain the 'BOOTSTRAP' file contains the file system that is loaded at system initialization.

While files created by the RIO Operating System or PROM debugger on the Development System or Micro Computer System are compatible, the bootstrap is not. Thus files may be interchanged between systems (procedure files are generally not transferable) but a system disk will bootstrap correctly only on the system for which it was designed.

1.2.3 SYSTEM INITIALIZATION

Where ZDOS is the primary file system, the bootstrapping of the operating system is from floppy disks. When a carriage return is entered as the first character after pressing RESET, or when the OS command is entered while in the Debug environment, the PROM monitor reads a 128 byte minibootstrap from track 17, sector 3 of the disk in drive 0. This program initiates a directory search on drive 0 for the files OS and ZDOS, which are then read into memory. Execution is started at the entry point of OS. This is one of two instances where a disk formatted as a system disk must be ready in drive 0. The other is when using the GET or SAVE commands of the PROM Debugger. In all other cases, while a particular drive search order may be implied, there is no difference in the utilization of drives.

This process is similar on systems which use DFS as the primary file system. The file 'BOOTSTRAP' contains the file system, which the disk controller loads directly from disk, using the standard disk search sequence of drive 1, drive 2, ..., drive 0. The PROM monitor then may communicate directly with the controller to load the file 'OS', again using the standard drive search sequence.

When execution of the file OS begins, an initialization procedure is performed that may or may not involve other files. A means is provided to read a set of commands from a file to extend this initialization process. In this way, a turnkey system can be implemented simply by editing the external initialization command file. Alternatively, the file OS can be edited directly to execute a user-defined command sequence at initialization time (see Appendix E). As part of the initialization process, memory is sized to determine the current configuration. If the sizing procedure determines the end of memory to be at other than a 4K boundary, a warning message is issued to indicate possible memory failure, thus providing a frequent diagnostic of system memory.

After initialization, OS responds with the message 'RIO REL v.cc (Where 'v' is the release version and 'cc' is the release cycle) followed by the system prompt character '%'. Any time RIO is ready to accept command input, this prompt character is printed.

1.2.4 COMMANDS

Command implementation is in one of two forms: for 'internal' commands, the code which actually implements the command is a part of the file OS and resides in memory when OS is loaded; 'external' commands are simply procedure-type files which are loaded into memory for execution. If a command is external, a search is made of all accessible directories for a file of the given name. In this context, the available command set is limited only by the particular files of procedure type which are on the 'ready' drives at a given moment. Therefore, user extension, modification, or replacement of the Zilog supplied software is a matter of file manipulation. For example, replacement of the file named OS on a system diskette with another file of the same name results in the automatic bootstrap of a user-defined software package. The majority of the standard RIO command set are implemented as external files. (Internal commands are noted as such in the command description, Chapter 5).

1.2.5 I/O

The I/O structure of RIO is designed to facilitate program development independent of physical device characteristics. To this end all I/O requests are made with reference to a 'logical unit' which may correspond to any of a given set of 'I/O devices'. In this way device modifications can occur with minimal impact on existing software.

The software required to control a particular hardware device or set of devices is termed the 'device handler' (used interchangeably with 'I/O device', 'I/O driver', 'device driver', or simply 'device'). Before a particular device can be accessed, its device handler must be loaded in memory. Initialization procedures may be required, and it may be desirable for the memory it utilizes to be protected from concurrent software routines. RIO provides command level control of these tasks and assumes that once this is done, the device is ready to handle I/O requests. This process is referred to as "activating" a device.

The fundamental concept underlying the RIO I/O structure is that of the 'logical unit' (also referred to as 'unit' or

'I/O unit') which enables I/O activity independent of a particular device. Units are 'defined' by linking or mapping a unit to a given device. I/O requests may not be made on undefined units, although some requests inherently result in unit definition.

Three units are predefined by RIO to handle console input (unit 1), console output (unit 2) and high volume printed output (unit 3). Unit 0 is used by system functions and is not available to the user. Units 4-20 (in the standard system) are available for user programming. Units 1, 2, and 3 have the mnemonics CONIN, CONOUT and SYSLST, respectively which can be used interchangeably with the literal unit designations, where applicable.

I/O requests are made with a standard vector format, containing information such as unit, data transfer address, data length, completion codes, and an optional supplemental parameter vector address. I/O requests are made by providing a pointer to the request vector (see below) and making a system call.

Note that programs which use the RIO I/O structure can remain unchanged so long as compatible I/O devices are provided. For instance, a BASIC system could immediately utilize a line printer by redefining SYSLST. No other software changes would be required.

CHAPTER 2

RIO EXECUTIVE

2.1 SYSTEM INITIALIZATION

As part of the system bootstrap procedure, the RIO Executive (OS) performs a series of initialization tasks. CON, the system console device, is initialized. The primary file system (or the master device - see section 3.1) is then initialized to identify the drives which are available. Memory size is determined by writing and subsequently reading a known pattern through memory until the comparison fails. If the last 'good' address is on other than a 4K boundary, i.e., nFFF, a warning message is generated to indicate possible memory failure. Memory occupied by PROM, OS, ZDOS, and CON is allocated. If the physical end of memory is other than FFFF, the nonexistent memory is also allocated (see section 2.3).

Initialization of the console device assigns default values to the line delete (LINDEL) and character delete (CHRDEL) symbols and the number of null characters (NULLCT) and line feeds (LFCNT) to be inserted after every carriage return. These values are NULLCT=1 (a single null character is sufficient for most CRT'S at speeds up to and including 19.2 Kbaud), LFCNT=1, LINDEL=7FH (rubout or delete), and CHRDEL=08H (control-H or backspace). The automatic line feed insertion mode (AUTOLF) and console character echo (ECHO) modes are set "on", and full duplex operation selected.

If the external initialization (EXTINI) bit (bit 2) of the system flag SYSFLG (see Appendices C and E) is set (=1), the external initialization command is executed. If the EXTINI bit is reset (=0) this initialization is not performed. Zilog-supplied software has this bit set; the external initialization command consists of 'DO OS.INIT'. This command causes the commands on file OS.INIT, default drive search sequence, to be executed as part of the initialization process.

The user may alter the external initialization bit using the PROM Debugger GET/SAVE commands (see section 5.13). See Appendix E for examples. In addition, the user can alter the initialization procedure by editing the contents of the file OS.INIT.

After possible redefinition of logical units by the commands on the external initialization file, the existing unit definitions are saved as the defaults. Subsequent unit definitions restoring a unit to its default will result in the unit definition existing at this point in the initialization process.

Concluding the bootstrap and initialization procedure, RIO prints an identifying message, the command prompt character is sent to the console output unit, and the system waits for command input.

2.2 FILE NAME CONVENTIONS

In the most general case, file names in RIO consist of three parts:

1) the device name specifying which device to search for the named file (e.g. $ZDOS);

2) the drive designation restricting the search to a particular element of the device (e.g. drive 2);

3) the file name itself.

The file name consists of from one to thirty-two characters, the first of which must be alphabetic. Subsequent characters may be alphanumeric ('A'...'Z' or '0'...'9'), or one of the special characters, question mark ('?'), underbar ('_'), or period ('.'). Upper and lower case characters are interpreted as entered, i.e., 'Status' is not the same as 'status'.

When a period ('.') is used within a file name, those characters in the name including and following the period are referred to as a file name 'extension'. For instance, the file name OS.INIT has the extension '.INIT', while the file name BOOK.CHAPTER.1 has the multiple extensions '.CHAPTER' and '.1'. The notion of file name extensions is

a useful convention for the user who wishes to categorize certain files by their names. Some programs such as the assembler or editor require that file names end with a particular extension--source files for the assembler must end in .S, while the editor creates a backup file with the extension .OLD--however, in general, RIO makes no distinction concerning extensions. In other words, a period is treated as any other valid character in a file name.

The drive designation consists of a single character from '0'...'7'. In the Zilog Development System, drive '0' is the right-hand drive, drive '1' is the left-hand drive. In the standard Zilog Micro Computer System, drive '0' is the right hand drive, drive '2' is the left-hand drive. The character '*' denotes a standard search sequence of drives '1', '2',...'7', '0'.

Device names are essentially file names prefixed with the character '$'. This character merely serves as a delimiter and is not really part of the name itself. In addition, the device name must have been made known to the system either by default initialization procedures or by the Activate command (see section 5.1). The devices known to OS after initialization are:

ZDOS	file system (where floppy diskettes are the primary file system media)
DFS	file system (where hard disks are the primary file system media)
CON	console driver
NULL	null operation device (see section 3.4.2)
PCON	PROM console driver
FLOPPY	PROM floppy disk controller interface
DISK	PROM hard disk controller interface

When constructing a file name, the character ':' is used to separate a device name from a drive designator and the character '/' is used to separate the drive designator or device name from the file name.

For example, the command STATUS may be entered as:

```
STATUS
/STATUS
0/STATUS
:0/STATUS
$ZDOS/STATUS
$ZDOS:0/STATUS
```

In the first case, the device name and drive designation are given default values. The default device is designated by the user to be the source of all 'unqualified' (no explicit device name) files and is termed the Master device. Default is ZDOS for floppy disk systems, and DFS for hard disk based systems, but may be redefined (and also displayed) by the MASTER command (see section 5.32). The default drive search order for command files is drive '0', followed by the standard search sequence (designated by drive '*') if the initial search of drive '0' is unsuccessful.

All 'qualified' file names (those with device or drive designations or the prefix '/') are treated as external commands. Thus, /DEBUG is not the same as the internal command DEBUG.

2.3 MEMORY MANAGEMENT

The RIO Executive includes a memory manager which controls allocation/deallocation for the system. A bit map is used to reflect the status of each 128 byte segment in the 65K address space. A set bit (=1) indicates that the segment in question is allocated. A reset bit (=0) indicates the segment is available for allocation. During system initialization, memory which is occupied by the system or which is nonexistent is marked as allocated. Subsequent memory utilization should be coordinated with information supplied by the memory manager, entry point MEMMGR, to avoid conflicting uses of the same memory segment.

2.3.1 MEMMGR

Subroutine calls to the system entry point MEMMGR can be used to allocate, deallocate, or determine the status of designated memory areas. Appendix I gives the details of these subroutine calls. Alternately, memory segments may be allocated, deallocated, or the current memory map displayed from the command level. Refer to Chapter 5 for details of the ALLOCATE, DEALLOCATE, and DISPLAY commands.

2.4 COMMAND STRING INTERPRETATION

Whenever RIO is ready to accept command input, the prompt character '%' is printed on the console output device. All characters entered (up to a maximum of 256, subject to device driver editing, see section 3.4.3) after the prompt character, up to and including the first ASCII carriage return (CR), are entered into the command string buffer. This input constitutes the command input string. The command separation character ';' is used to terminate a command but does not terminate command input. Thus, as many commands may be entered at one time as can be contained in the 256 byte command string buffer.

Several characters have special significance to the command string interpreter. As noted above, carriage return and semicolon are used to terminate commands and are therefore referred to as terminators. Space, horizontal tab (ASCII 09H), and left and right parentheses can separate command names from optional parameter lists and are referred to as delimiters.

There are two modes of providing system information to the user. In verbose mode, each command is echoed as it is extracted from the command string buffer. This is useful to verify input or when entering multiple commands per command string. In brief mode, commands are not echoed (except as entered).

After a command has been located in the command input string, an attempt is made to match it against a list of internal commands. In doing so, an internal command name may be abbreviated to the extent to which it is differentiable from other internal commands. For example,

the strings 'D', 'DE', 'DEB', 'DEBU', or 'DEBUG' all result in entering the PROM Debugger. If the abbreviation does not identify a unique internal command, then the first entry will be chosen. For example, 'D' and 'DE' refers to DEBUG rather than DEALLOCATE (Appendix H lists the internal command table contents in order). If no match is found, the command name is assumed to be the name of a file. The search order of drive '0', followed by drive '*', is then used in an attempt to open the file. If the file is located and is of type procedure, a request is made to the memory manager to allocate the space required to load the file. The values LOW_ADDRESS and HIGH_ADDRESS in the file's descriptor record define the memory which will be altered as a result of loading, and generally represent the lowest segment starting address and highest segment ending address, respectively. Note that since file I/O is not buffered, the latter is a function of the record size and may not equal the highest segment ending address. For example, loading a file consisting of 40 bytes linked at location 5000 having 80 byte records will affect memory locations 5000-507F rather than 5000-503F since a minimum of one record is required to contain the file. If the allocating request is successful, the file is loaded into memory.

After loading the file, two things may inhibit its execution. If it has a null entry point (=0) or if the delimiter following the command name is a comma, the command string interpreter suppresses command execution and instead processes the remainder of the command input string, if any. In this way, files may be loaded together and control passed to any one of them. For example, it may be desired that a user program and debugger be loaded with control passing to the debugger where instructions may be executed one at a time, breakpoints set, or registers given appropriate values prior to user program execution.

In the event command execution is not inhibited, a stack may be allocated consistent with the size specified in the descriptor record by LINK or IMAGE. (If a null (=0) stack size is requested, dispatch is made to the loaded file using the system stack space.) When several procedure files are loaded together, a stack is allocated for the first file in the command string with a nonnull stack size; no other stack space is allocated for the files loaded together. Two attempts are made to allocate the stack area. First the memory area following the loaded procedure to the end of memory is searched and, if unsuccessful, a second attempt is made, search-

ing from 0 to the beginning of the loaded file. If both attempts fail, no stack space is available and command execution cannot be initialized. Thus, normally the user stack is located immediately after the loaded file.

Since it is more efficient to not repeatedly load a command file which is to be executed several times in succession, RIO remembers the entry point of the last loaded file and provides the internal command XEQ (see section 5.42) to transfer program execution to that address. Most RIO commands can be executed repeatedly in this way.

Prior to executing the external command file, the memory map is examined to identify those segments which were allocated as a result of loading the file(s) to be executed. When return to OS is made, these segments will be deallocated. Note that in the event a file is loaded, but not executed, the space it occupies will be allocated until either explicitly deallocated (see the RELEASE command, section 5.35) or a return to OS is made from any external command file.

2.5 ERROR HANDLING

Wherever errors occur in the processing or execution of system commands, a message is directed to the console output device. Command processing then continues with the next command in the command string, if any.

2.6 PROGRAM EXECUTION OF COMMANDS

Any command or user program executable from the system console can also be executed from a program. This is accomplished by making a system call to RIO with reference to the command string to be executed. In this way programs can be chained together or complex overlay structures easily implemented. (See the Relocating Assembler and Linker manual for details of overlay creation.) System calls and the RIO system entry point are described in sections 3.2 and 4.5.

CHAPTER 3

I/O STRUCTURE

3.1 OVERVIEW

The I/O architecture of RIO is designed to a) facilitate user construction and implementation of device drivers to service the I/O requests of system or user programs; and b) simplify and standardize interface to all I/O drivers. To this end, all I/O requests are made to RIO with reference to a logical unit. RIO determines the proper routing for the referenced unit and passes control for servicing of the I/O request to the appropriate device driver.

The internal structures supporting this facility include the Active Device Table (ADT) and Logical File Table (LFT). The Active Device Table has one entry for every device known to the system at a given time and includes the device name and entry point. Devices are made 'known' to the system via the ACTIVATE command (see section 5.1), or they may be removed from the ADT by the DEACTIVATE command (see section 5.11). The current ADT contents can be reviewed with the LADT command (List Activate Device Table - see section 5.31).

Devices which are known to the system may be used to qualify a file name, thereby linking a logical unit to the named device. Unqualified file names (those without a device name prefix) are given a default routing to the master device (see MASTER, section 5.32).

The link between a logical unit and a specific device exists in the Logical File Table, each entry of which contains the address of the device that the corresponding unit is linked to. Before I/O requests may be processed via a logical unit, the unit must be defined. This unit definition may occur via the 'Assign' I/O request (see

Section 3.3) or it may occur as a result of the DEFINE command (see section 5.14).

As a part of system initialization units 1, 2, and 3 are defined as the console input, console output, and volume output devices, respectively, and are given the mnemonics CONIN, CONOUT, and SYSLST. Although these units are available for redefinition by the user, RIO assumes that these units represent the equivalent devices.

3.2 I/O REQUESTS - SYSTEM CALLS

I/O requests are accomplished by making a subroutine call to the RIO entry point SYSTEM (see Appendix C). The IY register must hold the address of a request vector, of the following format:

	Byte	Contents
IY ->	0	logical unit number
	1	request code
	2-3	data transfer address
	4-5	data length
	6-7	completion address
	8-9	error return address
	A	completion code
	B-C	supplemental parameter information

Logical Unit Number

The logical unit number is an integer in the range 1 to MAXLUN (20 in the standard system). Units 1, 2 and 3 are predefined by RIO to be console input, console output, and volume output.

Request Code

Identifies the operation requested.

Data Transfer Address

The memory address at which data movement will begin.

Data Length

Number of bytes of information in the transfer. This will be reset by the device to reflect the actual number of bytes transferred upon completion of the operation.

Completion Return Address

If bit 0 (least significant) of the request code is set (=1), those devices which are interrupt-driven will return control to the calling routine as soon as possible and continue the operation under interrupt control. At the time that the operation is completed, transfer will be made to the completion address which should exercise the responsibilities of an interrupt service routine (i.e., it must preserve all registers). However, an RETI instruction should not be executed, since lower-level interrupts are enabled by the interrupt handler. (If immediate return is desired, care must be taken not to change any words in the parameter vector, or use or change the data, until the operation is complete.) If return on completion is indicated (i.e., bit 0 is reset=0) the completion return address is ignored.

Error Return Address

If nonzero, the error return address will be used as the return address in the event of an error condition. The routine thus entered should execute an RET instruction after processing the error condition. Since the error condition is detected by the I/O driver, and the call to the error return address is made there, the programmer should not make assumptions about the elements on the stack above the return address.

Completion Code

The completion code is always set by the device and will indicate completion of the request and any errors. Error codes are universal (i.e., for all devices to which a given error applies, the error code is the same). Generally, bit 7=1 is used to signal operation complete, with bit 6=1 indicating an error condition (see Appendix A). All I/O devices must set this completion code prior to returning to the calling procedure.

Supplemental Parameter Information (Optional)

The two bytes of supplemental paramater information may be used to hold either additional data or an address to a vector supplement. The format of such an extension is defined by the device for a given request.

3.3 THE 'ASSIGN' I/O REQUEST

If a system call is made with the request byte equal to the 'ASSIGN' request code (02), the request is trapped by RIO for possible unit definition or supplemental parameter vector manipulation. The exact sequence is controlled by a set of flags in the first byte of the supplemental parameter vector.

If bit 7 (the most significant bit) of the flag byte is reset (=0), then RIO will format the supplemental parameter vector (see Appendix F), including the drive name, file name length, and file name fields, from information derived either from the string referenced by the Data Transfer Address of the request vector (bit 1=0) or from the string contained in the file name field (bit 1=1).

For example, suppose a user program requires one parameter which can be a qualified or unqualified file name. The user may elect to parse this parameter string in order to determine the device name (if any), drive designation (if any), file name and file length. Alternatively, a request vector can be set up with the Data Transfer Address field referencing the parameter string and the first (flag) byte

of the supplemental parameter vector with bits 7 and 1 reset to (this file name string must be terminated by a delimiter). RIO then moves the file name into the file name field of the supplemental parameter vector and sets the name length and drive designation. If no name is given, the name length field is set to zero. If no drive designation is given, the standard search sequence symbol '*' is put in the drive designation field. More importantly, the logical unit referenced in the request vector is linked to the device specified as part of the file name string, or the master device, if no device name is given.

As a second alternative, the user program can set bit 1 of the flag byte after moving (or assembling) the entire parameter string into the supplemental parameter vector file name field. RIO then formats the rest of the supplemental parameter vector in the same way as before.

If bit 7 of the flag byte is set (=1), the vector (and supplemental parameter vector) is assumed to be in a correct format, i.e., all fields hold valid information. If bit 0 is also set, the unit is linked to the master device. If bit 0 is reset, unit redefinition does not occur, maintaining the current unit-device link. In this last case, previous unit definition must have taken place. After the preceding steps are taken, the I/O request is passed to the intended device for processing. Subsequent I/O requests are routed directly to the device.

The following table summarizes the effects of specific supplemental parameter vector flag byte values during the 'ASSIGN' I/O request:

Flag Byte (hex)	Effect on ASSIGN I/O Request
0 (bit 1 reset)	RIO formats Supplemental Parameter Vector, Data Transfer Address is the address of the file name string
2 (bit 1 set)	RIO formats Supplemental Parameter Vector, file name field contains file name string

80 (bit 7 set and bit 0 reset)	RIO passes request directly to device (previous unit definition required)
81 (bit 7 set and bit 0 set)	RIO links unit to master device

3.4 STANDARD RIO I/O DEVICES

Five devices are known to RIO after default system initialization:

Device	Description
ZDOS or DFS	Primary file system
FLOPPY or DISK	Interface to device controller
NULL	Null device
PCON	PROM console driver
CON	System console driver

3.4.1 ZDOS

ZDOS is the file system for RIO on floppy disk based systems. It distinguishes between logical units and supports named files. Consult Chapter 6 for details of ZDOS request codes and request vector formats.

3.4.2 DFS

DFS is the ZDOS equivalent for hard disk based systems. Consult chapter 7 for details of DFS request codes and request vector formats.

3.4.3 NULL

NULL is a pseudo device driver which responds to all request codes. In most cases, the operation performed is, in fact, null--that is, no operation is performed.

Nonetheless, it responds with a completion code implying completion of operation.

I/O Request	Action
READ LINE READ BINARY	completion code = C9H (end of file), data length = 0
All others	completion code = 80H (operation completed)

This device provides a destination to which unwanted output can be diverted. It also provides a convenient way to check the integrity of a file. A file that can be copied to NULL has no record pointer errors, since a complete READ operation is performed. In the same way, all files on a disk can be copied to the NULL device with a single command, thus checking the file structure of the entire disk.

3.4.4 CON

CON is the default RIO console driver especially designed for CRT terminals. It is linked as part of the file OS which is loaded during system bootstrap. It allows the user to define the line and character delete symbols and supports arbitrary tab settings within a 134 character line length. The standard RIO I/O vector format is used in communicating with CON (see Appendix F).

During READ operations, entering the single character delete symbol (default = ASCII backspace, 08H) causes the last character placed in the buffer to be logically deleted. A backspace, space, backspace sequence is sent to the console to erase the character from the screen and reposition the cursor.

The line feed character (ASCII 0AH) forces the cursor to the start of the next line and places a space (ASCII 20H) in the buffer. This provides a convenient way to force the cursor or print mechanism to the beginning of the next line without terminating input. Note that no carriage return is placed in the buffer, i.e., input is logically a single line.

The line delete character (default = ASCII rubout or delete, 7FH) deletes from the console display and the input buffer all characters back to and including the previous carriage return. (If linefeeds or backspaces have been entered, not all of the displayed input string is erased from the display).

The input delete character (control-X) flushes the input buffer and echoes a backslash carriage return on the console display. The effect of the line delete character and the input delete character differs only when processing read binary requests (described below).

To input verbatim special characters (like rubout, control-X, etc.), an escape character (backslash) is provided. So, to enter 'AB<rubout>' type 'AB\<rubout>'. To enter '\', type '\\'. The backslash can be used to enter any character other than carriage return.

The console driver is not interrupt-driven nor does it distinguish between logical units. Modes can be set for linefeed insertion, number of nulls, and character echo. If in AUTOLF ON mode (default), and the value of LFCNT is nonzero, then LFCNT linefeeds are output to the console following every carriage return. After line feed insertion (if any), and if the value of NULLCT is nonzero, then NULLCT nulls (ASCII 0) are output to allow time for print head or cursor repositioning. Default is NULLCT=1, which is sufficient for CRT operation up to 19.2 Kbaud, and LFCNT=1. If in ECHO ON mode (default), then each character is echoed back to the terminal as it is read.

The ASCII tab character, control-I (09H), is expanded into an appropriate number of spaces only when it is output to a display device, thus compacting symbolic files where large numbers of spaces are required to improve readability. Tabs can be set by placing the cursor in the desired column and entering control-T (ASCII 14H) followed by 'T'. To clear a tab setting, the cursor is positioned in the column where the tab exists and the sequence control-T followed by a 'space' is entered.

The default tab settings are every eight columns, starting with the leftmost column as column 0. To change this default tab setting, the user may use the SET TABSIZE command (see 5.39).

Tabs can be altered in the file OS (from the PROM Debugger) to change the default tab settings of every eight columns. Different tabbing environments can be established and made into a file so that they may be altered by command (see Sections 5.37 and 5.38).

The following I/O requests are honored by CON:

INITIALIZE (00H) - Reads the current date into the default attributes table (see OPEN below) and sets default status area.

ASSIGN (02H) - Null operation returns operation complete

OPEN (04H) - If data length = 0: null operation. Otherwise up to 20 bytes can be requested from the default set of attributes including:

Type	20H (ASCII)
Record Count	0
Record Length	80H
Block Length	80H
Properties	0
Starting Address	0
Bytes in last record	0
Creation Date	Current Date

CLOSE (06H) - Null operation returns operation complete

READ BINARY (0AH) - Data length characters are received from the console. Entering a control-D (ASCII EOT, 04H) causes an end-of-file mark (FFH) to be placed in the buffer and the request terminated. Data length is reset to the actual number of characters read. The parity bit of each character is reset.

READ LINE (0CH) - A maximum of data length characters is received from the console up to and including the first carriage return. Data length is reset to the actual number of characters read. The parity bit of each character is reset.

WRITE BINARY (0EH) - Data length characters are sent to the console. An end-of-file mark (0FFH) results in termination of the request. Data length is reset to the actual number of characters written.

WRITE LINE (10H) - A maximum of data length characters is sent to the console up to and including the first carriage return. Data length is reset to the actual number of characters written.

READ STATUS (40H) - Transfers data length bytes of the CON status area to the area starting at the data transfer address. The CONSOL status flags are defined as follows:

Byte		
0	FLAG byte	Bit 0 Local Flag Bit 1 Auto linefeed insertion (AUTOLF) On=1 (default) Off=0 Bit 2 Echo On/Off On=1 (default) Off=0 Bit 3 Temporary Input buffer (TIB) Full=1 Empty=0 (default) Bit 4 Echo carriage return Off=1 On=0 (default) Bit 5 Escape pending Not pending=1 Pending=0 (default)

		Bit 6 Local flag Bit 7 Full/Half duplex Half=1 Full=0 (default)
1	reserved	
2	TIB	Holds last character which has been input from serial communication port but not yet transferred by a READ request.
3	Cursor Location	
4	reserved	
5..138	Tabbing Drum	134 positions used to mark tab settings (nonzero values)

WRITE STATUS (42H) - Transfers data length bytes from the data transfer address to the CON status area (see above).

DEACTIVATE (44H) - Null operation returns operation complete

READ ABSOLUTE (46H) - Data length bytes are received from the console device. Byte data is accepted exactly as transmitted. Data length is unaffected.

WRITE ABSOLUTE (48H) - Data length bytes are sent to the console device. Byte data is written exactly as given. Data length is unaffected.

All others - returns Invalid request completion code.

During write operations, entering a question mark causes the operation to pause until a second question mark is entered. Entering an ESCape always immediately terminates an I/O request.

3.4.5 PCON

The PROM console driver provides basic console I/O and becomes the default device for logical units 1 and 2 when OS encounters errors while requesting input from or output to these units. Refer to the MCZ or ZDS PROM User's Manual for full details.

3.4.6 FLOPPY

The PROM floppy disk driver is used by ZDOS as the access primitive for the floppy disk drives. Refer to the MCZ or ZDS PROM User's Manual for full details.

3.4.7 DISK

DISK is the hard disk controller interface provided for those utilities requiring access by sector address. Refer to Chapter 7 for full details.

CHAPTER 4

PROGRAM INTERFACE

4.1 PROGRAM LOCATION

The following table describes the memory utilization for the standard Microcomputer System (MCZ) and Development System (ZDS):

	MCZ 1/20	MCZ 1/35	ZDS (monito mode)
PROM	0-FFF	0-FFF	0-BFF
PROM Dedicated RAM	1000-13FF	1000-13FF	C00-FFF
RIO Executive (OS)	1400-24FF	1400-24FF	1000-20FF
Console Driver	2500-29FF	2500-29FF	2100-25FF
ZDOS	2A00-43FF	---	2600-3FFF
User Space	4400-	2A00-	4000-

RIO commands, and, in fact, all RIO procedure files, are written as subroutines. That is, the system return address is pushed on the stack when program execution of the procedure file begins. Command files are generally loaded into the low range of the program space for execution. Entry points and file sizes can be obtained using the EXTRACT or CAT commands (see Chapter 5).

The minimum requirement for program execution concurrent with RIO is that it be 'loadable' in the sense that the space required to read the file into memory be unallocated, and that sufficient space be available in the system to allocate a user stack.

The current state of the memory allocation map can be displayed using the DISPLAY command (see Section 5.19). In the MCZ 1/20 configuration, memory from 4400H is unallocated and is available for system or user command execution. The only concer of the user is to insure that all programs which coexist in memo form a disjoint memory space--i.e., if a program is to make syst calls which result in execution of external procedure

files, then all programs referenced which reside in memory concurrently must not occupy the same address space.

4.2 PARAMETER STRING ADDRESS

When the command string interpreter identifies an external file name and succeeds in loading the procedure file, the variable INPTR (see Appendix C) is given the address of the delimiter following the file name. Programs may alter the subsequent parameter string, if any, up to but not including the next terminator (carriage return or semicolon). Prior to program execution, this address is also pushed on the user stack, followed by the system return address.

4.3 PROGRAM STACK SPACE

To RIO, user programs look like subroutines. Before execution, the system stack pointer is saved, a user stack is allocated (if the requested stack size is not equal to zero), and the parameter string address and return address in RIO are pushed. Dispatch is then made to the program at its entry point. The stack size is determined by LINK or IMAGE, default being 80H bytes. Programs requiring larger stack space should be LINKed or IMAGEd with non-default stack sizes (ST=nn option).

4.4 PROGRAM TERMINATION - ERROR HANDLING

In addition, or as a supplement to internal error handling procedures, user programs may indicate certain errors to RIO by setting the variable ERCODE. In the RIO convention, if bit 6 is set, the value is taken to be an error code to be displayed. If the error code value is one of those which corresponds to a RIO error message (see Appendix A), then the message is printed instead of the value.

4.5 SYSTEM CALLS - SYSTEM ENTRY POINT

System calls for program execution of RIO procedure files is accomplished by making a subroutine call to the system entry point SYSTEM (see Appendix C) in the same way as an I/O request: The IY register must hold the address of a request vector of the following format:

Byte	Contents
IY -> 0	zero - indicates request is a system call rather than an I/O request
1	unused
2-3	command string address
4-7	unused
8-9	error handler address
A	completion code

Command String Address

Address of the first byte of command input string. This string is of indefinite length, but must terminate with a carriage return. The format for the command string is the same as if entered on the console input device (see section 2.3).

Error Handler Address

Address of the routine to which RIO jumps to handle error conditions. If zero, no jump will be made and error conditions will not be reported. This applies only to errors either generated within RIO or reported to it via ERCODE.

Completion Code

Either the completion code generated internally by RIO or the ERCODE reported by external file execution, if applicable, will be returned in this byte. Bit 6 set (=1) implies an error condition.

*** NOTE ***

When external files make system calls resulting in execution of other external files, the current state of the memory map needs to be saved in order to determine what space to deallocate as a result of program loading. This map is saved on the user stack occupying a block of 44H bytes. Care must be taken to allocate sufficient stack sizes for programs using this feature.

*** NOTE ***

RIO is not reentrant from its system entry point.

4.6 INTERRUPT STATUS

The initialization process associated with system restart sets interrupt mode 2, and the I register to the base address of the interrupt vector, with interrupts enabled. Proper system operation depends on this interrupt status. With this configuration the 8-bit vector supplied by the interrupting device is used with the contents of the I register to form a pointer to the interrupt service routine starting address. Zilog support devices can be programmed to supply appropriate interrupt vectors using this space. If program constraints make it necessary to alter the interrupt mode or I register, they must restore the proper conditions before making system calls that result in disk activity. See Appendix C for the system interrupt vector address for use in restoring the I register.

4.7 I/O UNIT UTILIZATION

The user is free to redefine all I/O logical units with the exception of '0'. Unit 0 is restricted to use by RIO. Units 1, 2 and 3 are predefined to be the console input device, console output device and system volume output device. Units 4 through 20 are initialized to be the master device. Redefinition of these units may result in abnormal system behavior upon return to RIO. This means that in the standard RIO configuration which allows 0-20, 17 units may be defined concurrently, with an additional 3 units predefined by RIO for use as console or line printer I/O devices.

4.8 PROGRAM EXAMPLES

In Appendix G are sample programs which the user is encouraged to edit, assemble, link, and execute. They illustrate some of the concepts introduced in previous sections of the chapter, including console I/O, parameter string processing, and file I/O.

CHAPTER 5

RIO COMMANDS

The following RIO commands are supplied as procedure type files which have the properties SECRET and WRITE PROTECTION. Each section is devoted to one command and has the following format:

SYNTAX

A description of the syntax of the command, giving parameter definitions, options and conventions. In all syntax descriptions, the following notation is used:

> Optional parts of a parameter list are enclosed in brackets, '[...]'.
>
> The symbol for logical or, '|', is used if either parameter separated by the symbol can be used, but not both.
>
> Parameters which can be repeated zero or more times are enclosed in parentheses and followed by an asterisk, i.e., (param)*.
>
> Parameters which can be repeated as necessary but must appear at least once are enclosed in parentheses and followed by a plus sign, i.e., (param)+.

Internal commands are indicated by notation just under the command name in the upper right hand corner of the page. In this case, the command as given in the syntax also indicates the extent to which the command may be abbreviated. Upper case characters are required while trailing lower case characters are not. The command may be entered in either abbreviated or unabbreviated form, in upper or lower case, e.g., 'DEB' is the same as 'd'.

DESCRIPTION

A general description of command operation and definition of options.

I/O UNIT UTILIZATION

The logical units that the command uses for a particular function. I/O error messages generally refer to the unit on which the error occurred.

EXAMPLES

Illustrative examples of command invocation, where appropriate.

Note that the length of the parameter string associated with any single command is intrinsically limited by the buffer space associated with the command string. This imposes a 256 character limit on commands entered via the console input device and a 512 character limit on commands created by the editor for execution as part of command files (see section 5.20). Of course, command files created by copying the console input device directly to a file are limited only by available memory when executing the command file.

SYNTAX

ACTIVATE device_name [address]

DESCRIPTION

Make a device known to the system by including it in the Active Device Table (ADT). It can thereafter be used as a device name in qualified file names. If the optional address is omitted, the file name referenced by the device_name will be located on the appropriate device and loaded if it is a device file (procedure type, subtype 1), has a non-null entry point, and does not overlay protected memory. The amount of memory allocated as a result of loading the file is kept as the SIZE field of the ADT entry for possible later use by the DEACTIVATE command. If the optional address is given, the file is assumed to have been previously loaded in memory. In this case the address parameter is taken as the device entry point. Since the memory bounds are unknown, the SIZE field of the ADT entry is set to a null value (0).

In either case, an Initialize I/O request is sent to the device to allow preparation for subsequent request handling.

I/O UNIT UTILIZATION

Unit 0: device file handling
Unit 2: error messages

EXAMPLES

ACTIVATE $MYDOS

locates file 'MYDOS' on the master device using the default drive search sequence. It is then loaded and given an Initialize request. An entry is created in the Active Device Table.

ACTIVATE $MYDOS:4/$MY.VIDEO.DRIVER

locates file 'MY.VIDEO.DRIVER' on device 'MYDOS', drive 4. It is then loaded, an Initialize request sent, and an Active Device Table entry created.

ACTIVATE $MY.PROM.DISK.DRIVER 0BFD

creates an Active Device Table entry using 0BFDH as the entry point. An Initialize request is generated.

5.2
ALLOCATE

ALLOCATE
(Internal Command)

SYNTAX

Allocate low_boundary high_boundary block_size

DESCRIPTION

Attempts to allocate block_size bytes (rounded up to a multiple of 80H bytes) of memory. The search begins at address low_boundary (rounded down modulo 80H), and the first block large enough and not extending beyond high_boundary (rounded up to the next multiple of 80H - 1) is marked as allocated in the system memory map. If allocation is not possible, the message INSUFFICIENT MEMORY is given.

I/O UNIT UTILIZATION

None

EXAMPLES

```
ALLOCATE 0 FFFF 120
```

starting at memory address 0, a search is made for a 180H byte (120H bytes rounded up to a multiple of 80H) memory segment.

```
ALLOCATE 5300  537F  80
```

attempts to allocate the single 80 byte memory segment starting at 05300H.

```
ALLOCATE 7400  8000  400
INSUFFICIENT MEMORY
```

no 400H byte block is available in the address range 7400-807FH.

ALLOCATE 8E00 9535 9535-8E00

use the expression evaluator to determine the blocksize to be allocated. (780H bytes).

BRIEF
(Internal Command)

SYNTAX

Brief

DESCRIPTION

Enters console Brief mode. Commands are not echoed on the console output device as interpreted and some command files suppress execution messages. See Verbose command.

I/O UNIT UTILIZATION

None

EXAMPLES

B

brief

5.4
CAT CAT

SYNTAX

```
CAT (match_string | T=type | P=props | D=drive |
    F=format | L=listing_disposition | DATE rel date |
    CDATE rel date)*
```

DESCRIPTION

Prints a catalog of entries in the file system directories which match the specified options. Given without options, all (non-secret) files in each active drive directory are listed. Options may be given in any order, and may appear more than once. Where options other than match_strings are specified more than once, the last one entered is used.

match_string

Fully- or partially-specified file names may be given, in which case only those directory entries which are identical to one of the fully specified file names or match one of the partially specified file names, are listed. Partially-specified refers to the use of the symbol '*' which denotes an arbitrary character string. For example, '*XYZ' matches any file which ends in 'XYZ'. 'ABC*XYZ' matches any name which starts with 'ABC' and ends with 'XYZ' but has any (or no) characters in the middle. The string '*' (which is equivalent to '**') matches any name. Match strings cannot be qualified file names, i.e., no device or drive name may be given (see the 'D=drive' option below).

T=type

Only files of the given type will be listed. Type must be one of 'D' (directory), 'A' (ASCII), 'B' (binary) or 'P' (procedure). Subtype may also be specified immediately following the type (e.g., 'P1' refers to files of procedure type, subtype 1). If no subtype is given, all subtypes of the specified type are listed.

P=props

Only files with exactly the specified properties will be listed. Props must be from 'W' (write protected), 'E' (erase protected), 'L' (properties locked), 'S' (secret), 'R' (random), 'F' (force memory allocation), or '&'. Use of '&' will allow any file with at least the specified properties to be listed. One or more properties may be concatenated in which case only files with exactly (or at least, if '&' is included) the specified properties will be listed.

D=drive

Only files from the specified drive will be listed. Drive must be from '0'...'7'. Default is to search directories from all ready drives on the master device. If a device is specified without a drive (i.e. D=$DFS), all ready drives on that device will be searched.

F=format

Specifies long (F=L) listing format. The short form (default) consists of name and drive while the long form gives name, drive, file type, record count, record length, file properties, starting address, date of creation, and date of last modification. Additionally, the number of files examined, the number of files listed, and the number of sectors used by listed files are given.

L=listing_disposition

The listing is normally routed to SYSLST but can be routed to any device or file. For example, L=$CON would route output to the console (SYSLST may also be defined as this device) or L=2/FILELISTING would route the output to file 'FILELISTING' on drive 2 of the master device. All output generated for the specified device or file will be buffered, i.e., several lines will be transferred at one time. While output is active at the console, entering a '?' character will cause output to stop until another '?' character is entered. If the ESCape character (ASCII 1BH) is typed, output will be terminated and control will return to the Executive. The '?' and ESCape features apply only to MCZ systems.

CDATE | DATE rel date

where rel is one of the relational operators '=', '>', '<', '>=', '<=', or '<>', and date is up to 6 digits or '*' representing a date to be compared against in 'yymmdd' form. '*' in a digit position specifies that that digit will be considered equal to anything. A date expressed with less than 6 digits is treated as being filled on the right with '*'s.
DATE refers to the date of last modification. CDATE refers to the date of creation. The entire option should be specified with no intervening blanks. For example:

```
CDATE>=7805
```

refers to all files created with dates in May of 1978 or later. This is equivalent to

```
CDATE>=7805**
```

If the referenced date field of the file descriptor has a character which is not a digit, it will not match unless that digit position of the match date has an '*' in it.

I/O UNIT UTILIZATION

Unit 2: error messages
Unit 3: default listing destination
Unit 4: directories
Unit 5: files listed in directory
Unit 6: non-default listing destination

EXAMPLES

```
CAT F=L
```

Lists all (non-secret) files (long format) from all ready drives.

```
CAT D=$DFS:2 P=W SYS*
```

Lists all files (short format) on drive 2 on the device $DFS (if it is ready) which are write protected (only) and whose names start with 'SYS'.

```
CAT F=L *.S *.L P=E& L=CAT.LIST
```

Lists all files (long format) which end with either '.S' or '.L' and are at least erase protected. Listing goes to file CAT.LIST on the master device.

```
CAT F=L P=& DATE>=780301
```

Lists all files (long format) with at least null properties that have been last modified on or since March 1, 1978.

5.5
CLOSE

CLOSE
(Internal Command)

SYNTAX

Close u/*

DESCRIPTION

Generates a Close (06) I/O request for (hexadecimal) logical unit 'u' or all logical units ('*'). Error returns are ignored.

I/O UNIT UTILIZATION

As noted above

EXAMPLES

CLOSE 5

generates a Close I/O request for logical unit 5.

CLOSE *

generates a Close I/O request for all logical units.

SYNTAX

```
COMPARE file_1 file_2
```

DESCRIPTION

Performs a comparison of the contents of file_1 and file_2 (excluding the descriptor record). If the file contents are identical, no message is given. For each byte comparison which fails, a message of the form

```
FILE1: BYTE 01FC RECORD 0003 = B6
FILE2: BYTE 01FC RECORD 0003 = A6
```

is given.

Pressing the escape key will terminate command execution. File_1 and file_2 may not be the same physical file, though they can be the same named file on different drives if the names are appropriately qualified.

I/O UNIT UTILIZATION

```
Unit 2:  error messages
Unit 6:  file_1 input
Unit 7:  file_2 input
```

EXAMPLES

```
COMPARE MYFILE YOURFILE
```

MYFILE and YOURFILE are read and compared. No message implies the files are identical.

```
COMPARE AFILE BFILE
I/O ERROR C9 ON UNIT 7
```

an end of file was reached on BFILE before the corresponding end of file on AFILE.

```
COMPARE 0/MYFILE 1/MYFILE
```

the two files have the same name, but they reside on different devices.

SYNTAX

COPY file_1 file_2 (A | U | O | RL=record length | T=type)*

DESCRIPTION

Copies file_1 (using a READ BINARY request) to file_2 (using a WRITE BINARY request). When the A (Append) option is specified, file_1 is copied (initially using a READ CURRENT request) to file_2 (initially using a WRITE CURRENT request) so that the first byte of file_1 is placed immediately after the last byte of file_2. The copy command then proceeds as before. Either file_1 or file_2 may be devices or fully qualified names. FILE attributes of file_1 are transferred to file_2. The options A (Append), U (Update), O (Output) are used to specify the type of open request performed on the destination file, file_2. The A (Append) option should only be used with ASCII files. See Chapter 6 for details. The default record length and type of file_2 will be the same as file_1. These attributes can be overridden by specifying one of 80, 100, 200, 400, 800 or 1000 for record length, or one of D (directory), B (binary), A (ASCII) or P (procedure) for type. In the event that the destination file or device is unable to support the record length attribute of the source file, the message

WARNING: RECORD LENGTH CHANGED

will be issued and the default attributes of the destination device will be used.

*** WARNING ***

If the record length of the destination file is not the same as the record length of the source file, either because the RL=record length option was specified or due to automatic record length modification (see above), the BYTECOUNT field in the source file's descriptor record is used to determine the number of bytes in the last record. In the event this value is incorrect, file truncation may result.

I/O UNIT UTILIZATION

Unit 2: error messages
Unit 6: source file
Unit 7: destination file

EXAMPLES

COPY MYFILE 2/MYFILE.TOO

Copies MYFILE on the master device, default drive search, to MYFILE.TOO on the master device, drive 2. File attributes of MYFILE are transferred to MYFILE.TOO. Error occurs if MYFILE.TOO already exists.

COPY $ZDOS:2/THE.FILE $MYDOS/THE.FILE RL=400 O

Copies THE.FILE on drive 2 of ZDOS to THE.FILE on device $MYDOS, default drive search. The record length of the destination file is 400H and its previous contents (if any) will be erased.

COPY ANOTHER.FILE $CON

Copies ANOTHER.FILE from the master device, default drive search to the device CON, the default RIO console device driver (see section 3.4.3).

COPY $CON 7/TEXT O

Copies from the device CON (see section 3.4.3) to the file TEXT on the master device, drive 7. If TEXT existed previously, its contents will be erased. Use a "control D" to exit the file created on $CON.

SYNTAX

COPY.DISK [s_drive TO d_drive] [V]

DESCRIPTION

Copies the disk in drive s_drive (default = 0) to the disk in drive d_drive (default = drive 2 (MCZ) or drive 1 (ZDS)). Before starting, the prompt message

DRIVES READY?

is given. Response other than 'Y' will abort the command. The disks are read and written directly (through the drive control ports) one track at a time; thus the previous contents, if any, are overwritten. It is not necessary that the destination disk be formatted. After the copy operation, a verification pass is made during which a track-by-track comparison is made. At the completion of this pass, the message

VERFICATION COMPLETE

indicates a successful verify operation. The message

*** 0016 VERIFICATION ERROR(S) ***

indicates the number of compare errors found during the verification attempt.

If the verification pass only is required, the 'V' option in the command line causes the copy cycle to be skipped.

I/O UNIT UTILIZATION

```
Unit 0:  FLOPPY or DFS interaction
Unit 1:  console interaction
Unit 2:  console interaction
```

EXAMPLES

```
COPY.DISK
DRIVES READY?Y
VERIFICATION COMPLETE
```

copies disk in drive 0 onto disk in drive 2 (for MCZ) or 1 (for ZDS).

```
COPY.DISK 3 TO 7  V
DRIVES READY?Y
VERIFICATION COMPLETE
```

verifies the disk in drive 3 is identical to disk in drive 7.

5.9
COPYSD COPYSD

SYNTAX

COPYSD file_name

DESCRIPTION

Copies a single ZDOS file from one diskette to another using a single disk drive. The file_name may be fully or partially qualified. The source and destination diskette are inserted as many times as necessary to copy the file. When either of the prompts:

INSERT SOURCE DISK. TYPE ANY KEY TO CONTINUE, ESCAPE TO ABORT:
INSERT DESTINATION DISK. TYPE ANY KEY TO CONTINUE, ESCAPE TO ABORT:

is sent to the console device, the user must place the source or destination diskette, as specified in the prompt, into the drive. If the user wishes to abort the command, the ESCape character (ASCII 1BH) is entered from the console device at this time; otherwise, any other character is entered. The file is created on the destination diskette with the same name and attributes that it has on the source diskette.

I/O UNIT UTILIZATION

Unit 1: console interaction
Unit 2: console interaction
Unit 6: ZDOS interaction

EXAMPLE

```
%COPYSD DATA.FILE
INSERT SOURCE DISK.  TYPE ANY KEY TO CONTINUE, ESCAPE TO ABORT:G
INSERT DESTINATION DISK.  TYPE ANY KEY TO CONTINUE, ESCAPE TO ABORT:
%
```

copies the file DATA.FILE from the source diskette to the destination diskette.

SYNTAX

DATE [yymmdd]

DESCRIPTION

Displays and optionally sets the date field which is used as the date of creation or date of last modification by ZDOS. Digits 'yy' specify the year, 'mm' the month, and 'dd' the day.

The Date command is part of the standard RIO external initialization command file OS.INIT. Editing the file manually is required to change the date set at system initialization.

Users are encouraged to maintain the DATE consistent with the actual date in order to maximize the utility of the information and capabilities provided by the file system(s).

I/O UNIT UTILIZATION

Unit 2: output response

EXAMPLES

```
DATE 780801
AUGUST 1, 1978
```

Sets and displays current system date.

```
DATE
AUGUST 1, 1978
```

Displays current system date.

SYNTAX

DEACTIVATE device_name

DESCRIPTION

Deletes device_name from the Active Device Table (ADT) and thus makes it unknown to RIO. A Close I/O request is generated for units linked to the deactivated device and a Deactivate I/O request is generated for the device itself. If the ADT size entry is non-null (> 0), the space allocated to the device handler is deallocated.

Deactivation is inhibited for the last active device, since there would be no source for further external commands and therefore no method to activate other device files. Likewise, the master device cannot be deactivated.

I/O UNIT UTILIZATION

Unit 2: error messages
Others: all units currently linked to device

EXAMPLE

DEACTIVATE $MYDOS

Removes MYDOS from Active Device Table and generates Close request for all units linked to MYDOS. A Deactivate request is sent to MYDOS and the space allocated to it deallocated (if ADT size entry > 0).

5.12
DEALLOCATE

DEALLOCATE
(Internal Command)

SYNTAX

DEAllocate block_address block_size

DESCRIPTION

Marks the block_size segment starting at block_address as unallocated in the system memory map. If any affected blocks were not previously allocated, the message

MEMORY PROTECTION

is given. Block_address will be rounded down and block_size will be rounded up to a multiple of 80 bytes.

I/O UNIT UTILIZATION

None

EXAMPLE

```
DEALLOCATE  5000  1400
```

deallocates the 1400 byte block starting at 5000 (must have been previously allocated).

```
DEALLOCATE  C000  180
MEMORY PROTECTION
```

the 180 byte block starting at C000 was not previously allocated.

DESCRIPTION:

Causes system to go into PROM Debugger Mode. The PROM Debugger provides certain commands and capabilities for debugging tasks. These commands are implemented as code segment overlays on systems with ZDOS as the primary file system. On systems using DFS as the primary file system, file system primitives are always available (because DFS executes on a separate processor), and code segment overlays are not required. **These commands require that the system diskette on Drive 0 must not be write-protected.**

SYNTAX:

DEBUG

EXAMPLE:

```
%D                      ;invoke Debug environment
>BREAK 5500             ;set a breakpoint
>Q                      ;return to RIO
%
```

PROM DEBUGGER COMMANDS:

- GET file_name

Loads a memory image into memory and stores its starting execution address in the user PC. A memory image file (file type=Procedure) can be created by the SAVE command or by the IMAGE or LINK commands. The files contain, as part of the descriptor, the starting address and length of one or more segments of contiguous memory that is stored in the file, as well as the starting address for execution of the file. The GET command can handle files of up to five segments of memory, but the segments may be of any size. **If more than six segments are in the file, only the first six (6) will be loaded.**

If any segment contains information within the range 0-13FFH, the system displays the message "MEMORY PROTECTION" and aborts the load attempt.

If any disk I/O errors occur, the system displays the message "FILE ERR code", where "code" is one of the operating system error codes.

- SAVE file_name (starting_address ending_address) + [E=entry] [RL=record_length]

The SAVE command copies segments of memory to diskette so that they may be restored later. Segments are saved beginning at the specified starting address, and terminating on a record boundary. In a case, therefore, where the starting address is 2000H, the ending address is 20FFH, and the record length is 80H, memory will be copied from the starting address to the ending address. But if the ending address is 2100H, then 101H bytes are being requested, and the next record must be completed, causing 2000 to 217F to be saved. Up to five such segments may be saved by repeating the start address and end address pair. The default record length for ZDOS-based systems is 80H, and the maximum record length is 400H. Other valid record sizes are 100H and 200H. The default record length for DFS-based systems is 200H. If no entry address is specified, the system assumes an entry address of 0.

If the entry address and record length parameters are both specified, they must be specified in the syntactical order specified above. If the system encounters a syntax error in the command line, it displays a Question-Mark character, and returns to Debug. Such syntactical errors might include incorrect or incorrectly entered file names or numbers, incorrect pairing of starting and ending addresses, failure to enter "RL=" or "E=", or failure to specify a value after "RL=" or "E=".

Any disk I/O error will cause the system to display the message "FILE ERR code", where "code" is one of the operating system error codes. One exception: if the diskette is full, the system may display the message "DISK FULL" rather than "FILE ERR D3."

*** USAGE NOTE ***

SAVE does not set the values LOW_ADDRESS, HIGH_ADDRESS, or STACK_SIZE in the file's descriptor record. These values must be set before the file can be loaded by RIO (see SET command description, Section 5.39).

*** WARNING ***

SAVE uses diskette allocation information retrieved directly from the diskette, whereas ZDOS uses a copy of the same information stored in memory. If ZDOS activities have immediately preceded the SAVE attempt, e.g., if the Debug environment is invoked by a Break from an operating system program, or will be returning immediately

after a SAVE, e.g., by a Quit return to the operating system without rebooting, the following procedures must be observed to insure that the diskette copy of the allocation information agrees with the memory copy, both before and after the SAVE operation.

1) If Debug was invoked via a breakpoint, and ZDOS is being accessed by the program in question, do not issue a SAVE command unless you can verify that all ZDOS files are closed.

2) When you are attempting a return to the operating system via a Quit command (after a SAVE has been done), immediately issue an Initialize command (I) to update the allocation maps.

This Change Page has been left blank intentionally.

5.14
DEFINE DEFINE

SYNTAX

```
DEFINE (unit file_name | unit device_name | unit * |
        *)+ [A | O | U | I | NF | NO]
```

DESCRIPTION

Links a logical unit (referenced by an integer from 1 to 20) to a currently active device or restores unit to the default established at system initialization (bootstrap). (Units 1,2,3 may be referenced by the mnemonics CONIN, CONOUT, and SYSLST, respectively.) If the unit was previously defined, a close request is generated. A file name may optionally be associated with the unit. Assign and Open requests may be generated for the unit.

Unit file_name

The unit is linked to the specified device if name is qualified, or to the master device if unqualified. An Assign request is sent to the device with file_name as a parameter, followed by an Open request (default open type is 'open for update').

Unit device_name

The unit is linked to device_name (must be active). No other I/O requests are generated.

Unit *

Links unit to the default established at system initialization.

*

Links all units to their defaults established at system initialization. The standard RIO system defaults to units 1, 2 and 3 defined as the console device; the remaining units are linked to the master device.

Options

A	open type = Append	(See Chapter 6 for
O	open type = Output	open types)
U	open type = Update	
I	open type = Input	
NF	open type = Newfile	
NO	Generate No Open request	

I/O UNIT UTILIZATION

Unit 2: error messages
Others: parameter dependent

EXAMPLES

DEFINE SYSLST $LPR

Defines the system volume output unit SYSLST (3) to be device driver LPR, which must be an active device. Subsequent I/O requests for unit 3 will be directed to this device.

DEFINE 6 MYFILE

Links unit 6 to the master device. Assign and Open (for update) I/O requests are then sent to unit 6 with MYFILE as parameter.

DEFINE *

Restores all units to the defaults established at system initialization (after bootstrap). All defined units are closed prior to being redefined as their defaults.

DEFINE 12 $YOUR.DOS/YOURFILE NO

Links unit 12 to the device YOUR.DOS (which must be active). An Assign I/O request is then generated with YOURFILE as the filename, but no Open request.

5.15
DELETE DELETE

SYNTAX

```
DELETE (match_string | T=type | P=props | D=drive |
        Q=query | DATE rel date | CDATE rel date)*
```

DESCRIPTION

Deallocates all records and deletes name from file directory of files which match the specified options. Given without option, all (non-secret) files in each active unit are deleted. Options may be given in any order, and may appear more than once. Where options other than match_strings are specified more than once, the last entered is used. As matches are made, if in query mode, a prompt is made to the console of the form:

```
DELETE drive/filename (Y/N/A/Q)?
```

One character is accepted as input and must be one of the following:

'Y'	Yes, delete the named file
'N'	No, do not delete the named file
'A'	Yes, delete the named file and all other files without further query.
'Q'	No, do not delete the named file and discontinue searching for file matches.

In general, if a file is listed with a given parameter list using CAT, it will be deleted using the same parameter list using DELETE.

match_string

Fully- or partially-specified file names may be given, in which case only those directory entries are deleted which are identical to one of the fully-specified file names or match one of the partially-specified file names. Partially-specified refers to the use of the symbol '*' which denotes an arbitrary character string. For example '*XYZ' matches any file which ends in 'XYZ'. 'ABC*XYZ' matches any name which starts with 'ABC' and ends with 'XYZ' but has any (or no) characters in the middle. The string '*' (which is equivalent to '**') matches any name. Match strings cannot be qualified file names, i.e., no device or drive name may be given (see the 'D=' option below).

T=type

Only files of the given type will be deleted. Type must be one of 'D' (directory), 'A' (ASCII), 'B' (binary) or 'P' (procedure). Subtype may also be specified immediately following the type (e.g. 'A0' refers to files of ASCII type, subtype 0). If no subtype is given, all subtypes of the specified type are deleted.

P=props

Only files with exactly the specified properties will be deleted. Props may include any of the following:

W	write protected
E	erase protected
L	properties locked
R	random
F	force memory allocation
&	files with at least the specified properties

One or more properties may be concatenated in which case only files with exactly (or at least, if '&' is included) the specified properties, will be deleted.

D=drive

Only files from the specified drive will be deleted. Drive must be from '0'...'7'. Default is to list all ready drives.

Q=query

Sets Query mode for delete operations. 'Q=Y' (default) causes a query before a delete; 'Q=N' suppresses queries. Note that the 'Q=Y' mode can be overridden by the query response 'A'.

CDATE | DATE rel date

where rel is one of the relational operators '=', '>', '<', '>=', '<=', or '<>', and date is up to 6 digits or '*' representing a date to be compared against in 'yymmdd' form. '*' in a digit position specifies that that digit will be considered equal to anything. A date expressed with less than 6 digits is treated as being filled on the right with '*'s.
DATE refers to the date of last modification. CDATE refers to the date of creation. The entire option should be specified with no intervening blanks. For example:

CDATE>=7805

refers to all files created with dates in May of 1978 or later. This is equivalent to

CDATE>=7805**

If the referenced date field of the file descriptor has a character which is not a digit, it will not match unless that digit position of the match date has an '*' in it.

I/O UNIT UTILIZATION

Unit 2: error messages
Unit 4: directories
Unit 5: files listed in directories

EXAMPLES

```
DELETE *.OLD Q=N
```

Deletes without query all files on all ready drives which end in '.OLD' (backup files are normally produced by the text editor).

```
DELETE T=A CDATE<773112
```

Deletes (after prompting) all ASCII type files created before December 31, 1977.

```
DELETE D=2 P=R *.BASIC Q=N
```

Deletes without query all random files on drive 2 whose name ends in '.BASIC'.

5.16

DISK.FORMAT DISK.FORMAT

DESCRIPTION:

Formats a hard disk cartridge or hard disk fixed platter, initializes the disk allocation map and utilization statistics in the superblock, and establishes the directory file.

SYNTAX:

DISK.FORMAT (S | D=drive | ID='diskname' | Q=query | N=number)*

Parameter	Description
S	Determines whether or not the disk is to be formatted as a disk from which to read the file system program into the controller. Such disks are referred to as system disks and contain the 'BOOTSTRAP' file and a pointer to that file within the superblock. If the "S" option is given in order to format a system disk, the file "BOOTSTRAP" is copied from the master device. This file implements the file system, and is read into memory when the controller is initialized.
D=drive	The drive number of the drive containing the disk to be formatted; specified as "drive." If the drive is not specified, the system issues the following prompt: DRIVE: Valid response: 0, 1, 2, 3, 4, 5, 6, or 7.
ID='diskname'	Up to 100 characters, not including a carriage return character, are written into the disk identification field in the superblock on the disk. The file system uses this field to verify that allocation maps in memory are for the disk being accessed. If no value is specified for this option, the system issues the following prompt: DISKID:

Parameter	Description
Q=query	Normally, before disk formatting begins, the system issues the following prompt: READY? Responses other than 'Y' or 'y' abort the format attempt. This verification may be bypassed by specifying 'Q=N'. The default is 'Q=Y'.
N=number	This option may be used to override the default directory size of 5 blocks. The directory will grow to accomodate more filenames, but the directory blocks will no longer be on the same track and seeks will have to be issued to look through the directory.

EXAMPLE:

```
%DISK.FORMAT S D=0 N=15 ID='RIO MCZ 1/35 SYSTEM DISK'
 READY?Y
```

formats the disk in drive 0 as a system disk with an initial directory size of 15 blocks.

USAGE:

The format program writes sector 0 so that the sector interlace map which maps logical records to disk addresses is valid. The format program then downloads a program which writes a known pattern on all sectors after sector 0. After this program completes, the following message is written to the console:

WRITE PASS DONE

The format program then downloads another program which will read back all sectors and build a free chain from which to allocate blocks to files. After this program completes, the following message is written to the console:

VERIFY PASS DONE

On a 5 megabyte platter, this will occur within approximately 5 minutes. If any blocks were read with an error, their record

numbers are written into an error block. The directory is then constructed. If the S option was specified, then the format program will attempt to copy the file 'BOOTSTRAP' onto the drive.

I/O UNIT UTILIZATION:

Unit 0: DISK interaction
Unit 1: console interaction
Unit 2: console interaction
Unit 4: DFS file system interaction

DESCRIPTION:

Attempts to recover lost file data due to software failure or abnormal program interruption. There are currently three levels of repair: Level 1, Level 2, and Level C (see USAGE, below).

SYNTAX:

DISK.REPAIR D=drive_number (L=level_indicator)

Parameter	Description
D=drive	The drive number of the drive containing the disk to be repaired; if no value is specified for this option, the system issues the following prompt: Drive: Valid response: 0, 1, 2, 3, 4, 5, 6, or 7.
L=level	The attempted repair level; if no value is specified for this option, the system issues the following prompt: Level: Valid response: 1, 2, or C.

EXAMPLES:

```
%DISK.REPAIR L=1 D=3
Drive number is 3, repair level is 1.
Ready? Y
6 sectors removed from free chain.
```

level 1 repair is administered to the disk in drive 3; 6 allocated sectors were removed from the unallocated sector list.

```
%DISK.REPAIR L=C
Drive: 0
Drive number is 0, repair level is C.
Ready?Y
Free sector count: 2576
```

DISK.REPAIR (continued)

USAGE:

- Level 1 Repair:

When software failure or external interruption prevents the cache of unallocated sectors maintained within the superblock from being updated, this level of repair will remove any allocated sectors from this cache. If the command does remove sectors, the message

"n sectors removed from free chain."

is sent to the console device, where n is the decimal number of sectors the command removed. If the state of the sectors is consistent with their being in the cache, the message:

"Free chain unmodified."

- Level 2 Repair:

Level 2 repair reconstructs the free chain from scratch. This level is useful for retrieving sectors that have been allocated, but are on the free chain and not in the unallocated sector list. Level 2 repair takes about seven minutes to run.

- Level C Repair:

Level C checks the free chain of unallocated sectors on the disk: it counts the number of free sectors and reports the count to the user, displays a warning for each unusable sector in the free chain, if any. If the number of free sectors listed by a DISK.STATUS significantly differs from the number of free sectors counted by Level C of DISK.REPAIR, it may be wise to perform a Level 2 repair on the disk. Level C requires about a minute to run.

Both Level C and Level 2 will take a very long time to run if there is more than one sector of unusable sector addresses, that is, if there are 130(?) or more unusable sectors on the disk.

I/O UNIT UTILIZATION:

Unit 0: disk sector access
Unit 2: execution messages

DESCRIPTION:

Lists statistics on how much of the DFS disk space on the specified drive has been used, and how much of it remains available for new file creation.

SYNTAX:

DISK STATUS [drive_number]

Parameter	Description
drive	The drive number of the drive containing the disk being status-checked; valid entries: 0, 1, 2, 3, 4, 5, 6, or 7.

EXAMPLE:

```
%DISK.STATUS 0

DRIVE 0    RIO MCZ 1/35 SYSTEM DISK
6638 SECTORS USED
3105 SECTORS AVAILABLE
```

USAGE:

Entering the command keyword without a qualifying drive number causes the system to respond by listing statistics for all ready drives. For example, DISK.STATUS issued on a system with one drive (two platters) causes the system to respond as follows:

```
%DISK.STATUS
             2 TRACK ERRORS
             0 CRC ERRORS
             0 SECTOR ERRORS
          2217 BUFFERS RECEIVED
         12404 BUFFERS SENT
          2217 MESSAGES RECEIVED CORRECTLY
             0 MESSAGES RECEIVED INCORRECTLY
             2 TIMES THAT COMMUNICATIONS LINK WAS RESET
          8312 I/O OPERATIONS PERFORMED
             0 UNFORMATTED WRITES
           330 FORMATTED WRITES
```

```
    0 UNFORMATTED READS
 7296 FORMATTED READS
    2 HEADER READS

DRIVE 0   790129
 7405 SECTORS USED
 2336 SECTORS AVAILABLE
    2 SECTORS UNUSABLE

DRIVE 1   CHECK IT OUT
 4327 SECTORS USED
 5414 SECTORS AVAILABLE
    2 SECTORS UNUSABLE
```

ERROR CONDITIONS:

If any sectors are unavailable due to read-back errors that occurred when the disk was formatted, the system issues the following message, where "n" is the number of unusable sectors:

```
n SECTORS AVAILABLE
```

The STATUS command uses data from the disk allocation maps on the disk to detect two error conditions. The number of free sectors and allocated sectors are added together and compared to the total number of sectors on the disk. If the number of free sectors and allocated sectors does not equal the total of sectors on the disk, the system issues the following message:

```
WARNING: DISK STATISTICS ARE INCONSISTENT
```

If the total number of sectors marked as unallocated in the sector map does not equal the free sector count, the system issues the following message:

```
WARNING: ALLOCATION IS INCONSISTENT
```

The second error condition is more serious than the first, because it is possible that sectors logically part of a file are marked as unallocated in the allocation map. These errors may result from memory failure, disk write failure, or an attempt at deleting files with pointer errors, etc. Generally speaking, either condition indicates a need to reformat the disk in question, but it may still be possible to read all files from the disk, and avoid loss of data.

I/O UNIT UTILIZATION:

Unit 2: Output Listing

SYNTAX

DISPLAY

DESCRIPTION

Displays the current state of the memory allocation map on the system console.

The memory allocation display is a matrix with one horizontal row for each 1000H bytes of memory. The point corresponding to each 80H byte segment of memory is either marked with 'A' if the segment is allocated, or '.' if it is free.

I/O UNIT UTILIZATION

Unit 2: memory display

5.20
DO DO

SYNTAX

DO command_file [parameter list]

DESCRIPTION

Executes commands from file command_file. The file is read into a dynamically allocated buffer. Each command line (as terminated by a carriage return) is then expanded according to the presence of certain expansion control symbols.

A 'parameter string' is a group of symbols delimited by either a blank (20H), comma (2CH), horizontal tab (09H), left parenthesis (28H), right parenthesis (29H), semicolon (3BH), or carriage return (0DH).

Simple parameter substitution is made for each occurrence of the string '#n', where n is an integer less than or equal to the number of parameters given. If n is greater than the number of parameters given, a Command Expansion Error is generated and processing is terminated. For each '#n', the nth parameter string from the parameter list is substituted. A maximum of 64 parameter strings may be passed in this manner. Parameters which are present but not referenced are ignored.

Conditional expansion of the command line can be controlled by the symbol pair '[' and ']'. At each occurrence of '[', the depth of conditional expansion increases by one. If the resultant depth is greater than the number of parameters given, the command line is scanned over until the matching ']' is located. If the resultant depth is not greater than the number of parameters given, the '[' is deleted from the command string and expansion continued. At each occurrence of ']', the depth of conditional expansion decreases by one. Thus, the command string 'AB[C]DE' would expand into 'ABCDE' only if at least one parameter were given. Otherwise, the resultant command string (after expansion) would be 'ABDE'. Note that parameters may control command string expansion regardless

of whether or not they are used in that expansion.

After command string editing according to the above rules, a system call is generated to execute the resultant command string.

This command is reentrant (and has the 'force allocation' property) and may call itself up to a depth limited only by the amount of memory available. That is to say, FILE.X may contain the command 'DO FILE.Y', which may contain the command 'DO FILE.Z', etc., down to a level where insufficient memory exists to allocate buffer space. For short command files (a few records) in a 32K system, this depth is approximately 15, depending on memory requirements for other command executions.

*** WARNING ***

Due to the force allocation property of this command, it will over-write the memory where it is loaded, whether or not the memory is all ready allocated. This command should be linked to load at an address that will not affect memory that is preallocated for system use.

I/O UNIT UTILIZATION

Unit 0: command file
Unit 2: error messages

EXAMPLES

The examples have the following format:

FILENAME:	DO file contents
Command:	Command line entered
Result:	The expanded file contents which are to be executed.

1) PRINT: ACTIVATE $LPTR;COPY #1 $LPTR;
DEACTIVATE $LPTR

Command: DO PRINT MYFILE

Result: ACTIVATE $LPTR
COPY MYFILE $LPTR
DEACTIVATE $LPTR

Failing to give a parameter would result in a Parameter Expansion Error.

2) PRINT: ACTIVATE $LPTR
[COPY #1 $LPTR[;COPY #2 $LPTR[;COPY #3 $LPTR]]]
DEACTIVATE $LPTR

Command: DO PRINT MYFILE

Result: ACTIVATE $LPTR
COPY MYFILE $LPTR
DEACTIVATE $LPTR

Simple parameter substitution is performed for as many strings enclosed within brackets as there are parameters given. Only the parameter 'MYFILE' was present, therefore the command after (and including) the second '[' was ignored.

Command: DO PRINT FILE1 FILE2

Result: ACTIVATE $LPTR
COPY FILE1 $LPTR;COPY FILE2 $LPTR
DEACTIVATE $LPTR

Two parameters were given so that two levels of conditional expansion were valid.

3) BATCH: EDIT #1[;ASM #1[;LINK $=4400 #1[;#1]]]

Command: DO BATCH MYFILE A L X

Result: EDIT MYFILE;ASM MYFILE;LINK $=4400 MYFILE;MYFIL

Command: DO BATCH MYFILE A

Result: EDIT MYFILE;ASM MYFILE

In the first command, four parameters were given indicating conditional expansion of four levels. In the second command, only two were given, limiting expansion to two levels. The characters 'A', 'L', and 'X' could have been any character string although they serve as symbolic notations for assemble, link, and execute. This is an example where the number of parameters controls expansion but the parameters themselves do not take part in the expansion.

SYNTAX

```
DUMP file_name [m[ n]]
```

DESCRIPTION

Converts the referenced file into a hexadecimal/ASCII dump on unit SYSLST. Each byte of the file is displayed in hexadecimal. In addition, printable characters are displayed as ASCII symbols, while unprintable characters are displayed as '.'.

If m and n are specified, the dump starts with record m and continues through record n. If m and/or n are unspecified, the dump starts with the first and continues through the last, respectively.

While output is active at the console, entering a '?' will cause output to stop until another '?' is entered. If the ESCape character (1BH) is entered, output will be terminated.

I/O UNIT UTILIZATION

```
Unit 3:  output listing
Unit 4:  file to be dumped
```

EXAMPLES

```
DUMP $MICRO.80:2/DATA
```

Dumps the file 'DATA' from device MICRO.80, drive 2 on the system volume output unit.

5.22
ECHO ECHO

SYNTAX

ECHO string

DESCRIPTION

Copies the string following the command name up to, but not including, the command terminator, to the console output device. This provides a method to send messages to the console from the command line.

I/O UNIT UTILIZATION

unit 2: string output

EXAMPLES

ASM MYFILE;LINK $=4400 MYFILE;ECHO <control-G>

This would send a control-G (bell) to the console output device after completion of the assembly and link.

ECHO is also useful to provide instructions to the user of the console command file. For example,

COPY,;ECHO INSERT DISKETTES;PAUSE;I;X 4400 #1 #2

can be used to copy files from one diskette to another, neither of which have the command file COPY on them.

SYNTAX

ERROR [error_code|*]

DESCRIPTION

Prints the meaning of error_code when returned by RIO or a device as a completion code. If the optional error_code is '*', all error_code meanings are displayed. If error_code is omitted, this description of the ERROR command is printed.

I/O UNIT UTILIZATION

Unit 2: output

EXAMPLE

```
%ERROR 43
43: MEMORY PROTECT VIOLATION
```

SYNTAX

ERRORS

DESCRIPTION

Prints a summary of the recoverable disk errors which have occured since system bootstrap. Output is of the form:

```
THE FOLLOWING RECOVERABLE ERRORS HAVE OCCURRED SINCE SYSTEM RESTART
        0000 SEEK ERRORS
        0000 SECTOR ADDRESS ERRORS
        0000 DATA TRANSFER ERRORS
```

Reference the Z80-MCZ PROM User's Manual for a detailed explanation of these errors.

I/O UNIT UTILIZATION

Unit 2: output listing

*** NOTE ***

This command is implemented only with the MCZ 1/20 PROM date coded 78089 or later, or with the MCZ 1/35 PROM date coded 780529 or later when ZDOS is used as a secondary file system.

SYNTAX

```
EXTRACT file_name
```

DESCRIPTION

Lists record count, record length, and the number of bytes in the last record of file_name. If the file is of type procedure, the file entry point, the lowest and highest memory addresses affected by the file, and the addresses of the memory segments which make up the file are also displayed. For files created by IMAGE, the segment addresses are those given in the parameter list. However, LINK provides an optimizing algorithm for segment allocation dependent on program memory utilization and file record length. Thus, EXTRACT can be used to determine the best record length for a procedure file.

I/O UNIT UTILIZATION

```
Unit 0:  file input
Unit 2:  output listing, error messages
```

EXAMPLE

```
%EXTRACT EXTRACT
RECORD COUNT = 0001  RECORD LENGTH = 0400
      NO. OF BYTES IN LAST RECORD = 0400
ENTRY POINT = 4400  LOW ADDRESS = 4400  HIGH ADDRESS = 47FF
      STACK SIZE = 0080
SEGMENTS:
4400 45F2
```

FORCE
(Internal Command)

SYNTAX

Force command parameter_list

DESCRIPTION

Causes all command files in the current command string to be loaded regardless of previous memory allocation. Normally, a procedure file will be loaded only if the memory space it requires is unallocated. Sometimes it is convenient to load a file into previously allocated memory space. Command overlays or recursive program calls are two examples. As an alternative to using the Force command, the properties of a file can include F (force memory allocation), which has the same effect as the Force command, but only for that file.

I/O UNIT UTILIZATION

None

EXAMPLES

FORCE DISPLAY

loads and executes the procedure file DISPLAY even if the memory space it requires is preallocated.

F FILEA,FILEB,;FILEC

loads the procedure files FILEA and FILEB, but does not execute either. FILEC will be loaded (and executed) only if the memory it requires is available; i.e., the context of the FORCE does not extend into subsequent commands.

SYNTAX

```
FORMAT (S | D=drive | ID='diskname' | Q=query)*
```

DESCRIPTION

Formats a diskette into 77 tracks of 32 sectors, initializes the disk allocation map and disk utilization statistics. An empty (except for one entry for itself) directory file is established.

Thirteen sectors are allocated for the disk allocation map (3 sectors) and directory (10 sectors). When a system disk is formatted, an additional 64 sectors (2 tracks) are preallocated for the RIO bootstrap and RIO Debug Get/Save package.

The parameter list specifies the following options:

S

Determines whether or not the disk is to be formatted as a system disk. Systems disks have dedicated areas for the bootstrap and GET/SAVE overlays.

When the 'S' option is given and the disk is being formatted on any drive except 0, the bootstrap and GET/SAVE overlays are read from the disk in drive 0. If the 'S' option is given and the disk is being formatted on drive 0, a prompt is made for the user to insert a formatted system disk in drive 0 replacing the disk being formatted. After this is done, entering any key will cause the system bootstrap and the GET/SAVE overlays to be read into memory and another prompt to be issued. Again entering any key will result in the bootstrap and the GET/SAVE overlays being written onto the disk, thereby making it a 'system' disk.

D=drive

The drive containing the disk to be formatted is given as 'drive'. If the option is not present, the query

DRIVE:

will be given, the correct response to which must be an integer 0...7.

ID='diskname'

Up to 24 characters not including a carriage return are used to identify the disk. If a single quote is to be part of the new name it must be immediately preceded by a percent (%) sign. These are written on the disk and used by ZDOS to determine disk allocation map validity. If this option is not given, the query

DISK ID:

will be given.

Q=Query

Normally, before formatting commences, the query

READY?

is given. Any response other than 'Y' will result in aborting the format. The generation of this query may be inhibited by giving the 'Q=N' option. The 'Q=Y' option is the default and has no effect.

RELEASE
(Internal Command)

SYNTAX

Release

DESCRIPTION

As mentioned in Section 4.1, memory required for procedure file loading is allocated immediately preceding execution, and deallocated after program completion. In the case where a file is loaded but no external file is executed (for example, after examination with the Debugger), it may be necessary to deallocate the space it occupies. This command deallocates any memory allocated as a result of procedure file loading since the last execution of an external command.

I/O UNIT UTILIZATION

None

EXAMPLE

```
%MOVE,
%STATUS
MEMORY PROTECT VIOLATION
%R
%STATUS

DRIVE 0      RIO.MCZ.SYSTEM.DISK
659 SECTORS USED
1805 SECTORS AVAILABLE
```

SYNTAX

RENAME (oldfile newfile | device:drive ID='new_disk_name')*

DESCRIPTION

For each sequence, "oldfile newfile", changes the name of "oldfile" to "newfile" on the disk drive specified by oldfile. If in Verbose mode, the following message will be printed for each name change.

oldfile--->newfile

For each sequence, "device:drive ID='new_disk_name'", the name of the disk in the specified drive is changed to "new_disk_name". The device must be either $FLOPPY for diskettes or $DISK for hard disks; no pseudonyms for these devices may be used. The "new_disk_name" may be up to 24 characters for $FLOPPY or 100 characters for $DISK, and may include any character except carriage return or semicolon. If a single quote is to be part of the new name, it must be immediately preceded by a percent sign (%). The disk is renamed by first initializing the allocation maps, reading directly from the disk (via the floppy or hard disk driver) the map sector on which the disk ID is saved, altering it, and rewriting the sector. A second initialization is then made to update the disk name in memory.

I/O UNIT UTILIZATION

Unit 0: file I/O

NOTE: Diskettes formatted on OS 2.1 software are NOT compatible with RIO. Refer to Appendix D for conversion details.

Format interacts with FLOPPY through the system. The driver $FLOPPY must appear in the Active Device Table. On floppy disk-based systems, this occurs automatically. On hard disk-based systems, FLOPPY is part of ZDOS, and has an entry point two greater than the entry address for ZDOS. It must be activated separately. For example, if the ZDOS.60 is being used on a 64K system, has been renamed to ZDOS, and has an entry point of E000H, than a sequence of commands such as:

```
ACTIVATE $ZDOS; X* $FLOPPY E002
```

would have to have been executed at some time prior to issuing a FORMAT command.

I/O UNIT UTILIZATION

Unit 0: FLOPPY interaction
Unit 1: console interaction
Unit 2: console interaction
Unit 4: ZDOS interaction

SYNTAX

```
HELP (key_word|'*')*
```

DESCRIPTION

Prints a description of key_word(s). If the final argument is *, a list of valid key_words that can be used as further modifiers of the preceding key_words is displayed. If the final argument is omitted, a general description of the use of the preceding modifying key_words is printed.

I/O UNIT UTILIZATION

```
Unit 2:  Help message
Unit 4:  File I/O
```

EXAMPLES

```
%HELP *
```

prints a list of all initial key_words for which there is HELP.

```
%HELP DELETE
```

prints a description of the RIO 'DELETE' command.

SYNTAX

```
IMAGE file_name (first_location last_location)+
                [E=entry point] [RL=record length]
                [ST=stack size]
```

DESCRIPTION

Copies memory images to a specified file. The resultant file will be procedure type, subtype 0. The first and last locations of each memory segment, optional entry point address (default=0), record length (80H, 100H, 200H, 400H, 800H, or 1000H; default=80H bytes) and stack size (default=80H bytes) are given in hexadecimal. At least one but no more than 16 segments may be specified. When writing the file, the exact memory locations, including first_location and last_location, are copied for each segment. The lowest and highest memory addresses referenced by the file are saved in the descriptor record (refer to Appendix J) and are used by the RIO Executive when requesting memory allocation prior to loading.

I/O UNIT UTILIZATION

```
Unit 0:  file I/O
Unit 2:  error messages
```

EXAMPLE

```
IMAGE TWO.BLOCKS  4400  4425  7000  7FF0  E=7000
```

Copies contents of memory locations 4400 to 4425 and 7000 to 7FF0 to file TWO.BLOCKS. The file will contain 33 records of 80H bytes each, with an entry point = 7000 and stack size = 80H.

INITIALIZE
(Internal Command)

SYNTAX

```
Initialize [device_name [parameter list]]
```

DESCRIPTION

Sends an Initialize request to the master device or to the optionally specified device (which must be active). Result is device dependent. The supplemental parameter address of the vector points to the delimiter after the command or device_name, if given.

I/O UNIT UTILIZATION

Unit 0: I/O request

EXAMPLES

```
INIT $MY.VIDEO.DRIVER  BUFFER = C000
```

Sends Initialize request to MY.VIDEO.DRIVER, with a pointer to the space preceding 'BUFFER'.

```
I
```

Sends Initialize request to master device.

5.31
LADT LADT

SYNTAX

LADT

DESCRIPTION

Lists the currently active devices, their entry points, size, and which logical units are linked to each.

A size of zero implies that the device is in PROM or was activated with a preloaded entry point given. In either case, no memory is deallocated upon deactivation.

I/O UNIT UTILIZATION

Unit 3: listing output

EXAMPLE

```
LADT

DEVICE          ADDRESS     SIZE     UNITS

ZDOS            2A00        1A00      0  4  5  6  7  8  9 10 11
                                     12 13 14 15 16 17 18 19
CON             252B        0500      1  2  3
NULL            214D        0000     20
PCON            0BE8        0000
FLOPPY          0BFD        0000
```

SYNTAX

MASTER [device_name]

DESCRIPTION

Displays the current master device or, optionally, makes another currently active device the default source for unqualified files. This provides the user with the potential to easily utilize multiple file systems concurrently without the burden of always fully specifying file names.

I/O UNIT UTILIZATION

Unit 2: error messages

EXAMPLES

MASTER $NEW.DOS

Makes NEW.DOS the default device for unqualified file names.

```
MASTER
NEW.DOS IS THE MASTER DEVICE
```

5.33
MOVE MOVE

SYNTAX

```
MOVE (match_string | T=type | P=props | F=format |
     D=destination_device | S=source_device |
     L=listing_disposition | Q=query | DATE rel date |
     CDATE rel date)*
```

DESCRIPTION

The directory on the source device is searched and files which match the specified option are copied from the source device to the destination device. Default destination and source devices for MCZ are master device, drive 2, and master device, drive 0, respectively. For the Development System, the defaults are drive 1 and drive 0, respectively.

match_string

Fully or partially specified file names may be given, in which case only those directory entries which are identical to one of the fully-specified file names or match one of the partially-specified file names are moved. Partially-specified refers to the use of the symbol '*' which denotes an arbitrary character string. For example, '*XYZ' matches any file which ends in 'XYZ'. 'ABC*XYZ' matches any name which starts with 'ABC' and ends with 'XYZ' but has any (or no) characters in the middle. The string '*' (which is equivalent to '**') matches any name. Match strings cannot be qualified file names, i.e., no device or drive name may be given.

T=type

Only files of the given type will be moved. Type must be one of 'D' (directory), 'A' (ASCII), 'B' (binary) or 'P' (procedure). Subtype may also be specified immediately following the type (e.g. 'PO' refers to files of prodecure type, subtype 0). If no subtype is given, all subtypes of the specified type are moved.

P=props

Only files with exactly the specified properties will be moved. Props must be from 'W' (write protected), 'E' (erase protected), 'L' (properties protected), 'S' (secret), 'R' (random), 'F' (force memory allocation), or '&'. Use of the '&' will allow any file with at least the specified properties to be moved. One or more properties may be concatenated, in which case only files with exactly (or at least, if '&' is included) the specified properties will be moved.

D=destination_device

Defines device to which files are copied. Any active device name or drive designation may be given. Only device name and drive name are relevant. File names are ignored. Default is drive 2, master device (MCZ), or drive 1, master device (ZDS).

S=source_device

Defines device from which files are copied. Any active device name or drive designation may be given. Only device name and drive name are relevant. File names are ignored. Default is drive 0, master device.

F=format

Specifies long (F=L) or short (F=S) listing format. The short form (default) consists of name and drive while the long form gives name, drive, file type, record count, record length, file properties, starting address, date of creation, and date of last modification. Additionally, the number of files examined, the number of files moved, and the number of sectors used by moved files is also given.

L=listing_disposition

The listing is normally routed to SYSLST but can be routed to any device or file. For example, L=$CON would route output to the console (SYSLST may also be asigned to this device) or L=2/FILELISTING would route the output to file 'FILELISTING' on unit 2 of the master device. All output generated to the specified device or file will be buffered, i.e., several lines will be transferred at one time. While output is active at the console, entering a '?' character will cause output to stop until another '?' character is entered. If the ESCape character (ASCII 1BH) is typed, output will be terminated and control will return to the Executive.

Q=query

Permits selective copying of files that match the other options. Default is Q=N, which moves all files which match the given criteria. If Q=Y is specified, a message of the form:

```
MOVE source_device : source_drive / source_file name TO
      destination_device : destination_drive / destination_filename
      (Y/N/A/Q)?
```

response of 'Y(es)' will move the file, 'N(o)' will not move it and go on to the next file, 'A(ll)' will move the file and suppress the query for all subsequent files, and 'Q(uit)' will not move the file and will terminate the program. Any other response will cause the query to be repeated.

CDATE | DATE rel date

where rel is one of the relational operators '=', '>', '<', '>=', '<=', or '<>', and date is up to 6 digits or '*' representing a date to be compared against in 'yymmdd' form. '*' in a digit position specifies that that digit will be considered equal to anything. A date expressed with less than 6 digits is treated as being filled on the right with '*'s.
DATE refers to the date of last modification. CDATE refers to the date of creation. The entire option should be specified with no intervening blanks. For example:

```
CDATE>=7805
```

refers to all files created with dates in May of 1978 or later. This is equivalent to

```
CDATE>=7805**
```

If the referenced date field of the file descriptor has a character which is not a digit, it will not match unless that digit position of the match date has an '*' in it.

I/O UNIT UTILIZATION

```
Unit 0:  directory
Unit 3:  default listing destination
Unit 4:  source file
Unit 5:  destination file
Unit 6:  non-default listing destination
```

EXAMPLES

```
MOVE D=$NULL F=L P=&
```

Will copy all files from (default) drive 0 to the Null device and print a long format list including the number of files moved and the number of sectors they occupy. This is a convenient way to check the integrity of each file on a disk.

```
MOVE T=P SYS* L=$LPRINTER S=2 D=0 CDATE<780915
```

Will copy all procedure files whose names start with 'SYS' that were created before September 15, 1978, from the master device, drive 2, to the master device, drive 0. The listing will be sent to the device LPRINTER.

5.34
PAUSE PAUSE

SYNTAX

PAUSE

DESCRIPTION

Issues successive Read Status requests to unit 1 (CONIN) until either the ESCape Pending flag or the TIB Full flag is active (see section 3.4.3). If a character is ready to be input, it is absorbed and the program executes a normal return. If an ESCape is pending, subsequent commands in the command string are ignored.

I/O UNIT UTILIZATION

Unit 1: read request

EXAMPLE

The content of the command file MOVE.IT is:

```
MOVE,;ECHO INSERT DISKETTES;PAUSE;I;X 4400
```

This command file will result in the following interaction when executed (Brief mode):

```
DO MOVE.IT                ;MOVE is loaded
INSERT DISKETTES          ;ECHO is loaded and executed
                          ;PAUSE waits for one character
                          ;(not ESC) to be entered and
                          ;then MOVE is executed (MCZ
                          ;address). Entering ESC would
                          ;have resulted in direct
                          ;return to RIO without executing
                          ;MOVE
```

DESCRIPTION:

Copies an ASCII-type file to the SYSLST unit. The specified filename may be fully or partially qualified. Entering the Escape character keystroke (ESC: ASCII 1BH) at the terminal terminates output.

SYNTAX:

PRINT file_name

Parameter	Description
file_name	The name of the user file; an alphanumeric character string, excluding special characters; upper or lower case (the command keyword must be in upper case characters).

EXAMPLE:

%PRINT REPORT.FILE

Prints the file named REPORT.FILE on SYSLST.

%PRINT 0/MYFILE

Copies the file named MYFILE located on Drive 0 of the Master device to SYSLST.

I/O UNIT UTILIZATION:

Unit 2: error messages
Unit 3: file listing

This Change Page has been left blank intentionally.

RELEASE
(Internal Command)

SYNTAX

Release

DESCRIPTION

As mentioned in Section 4.1, memory required for procedure file loading is allocated immediately preceding execution, and deallocated after program completion. In the case where a file is loaded but no external file is executed (for example, after examination with the Debugger), it may be necessary to deallocate the space it occupies. This command deallocates any memory allocated as a result of procedure file loading since the last execution of an external command.

I/O UNIT UTILIZATION

None

EXAMPLE

```
%MOVE,
%STATUS
MEMORY PROTECT VIOLATION
%R
%STATUS

DRIVE 0   RIO.MCZ.SYSTEM.DISK
659 SECTORS USED
1805 SECTORS AVAILABLE
```

5.36
RENAME RENAME

SYNTAX

RENAME (oldfile newfile | device:drive ID='new_disk_name')*

DESCRIPTION

For each sequence, "oldfile newfile", changes the name of "oldfile" to "newfile" on the disk drive specified by oldfile. If in Verbose mode, the following message will be printed for each name change.

oldfile--->newfile

For each sequence, "device:drive ID='new_disk_name'", the name of the disk in the specified drive is changed to "new_disk_name". The device must be either $FLOPPY for diskettes or $DISK for hard disks; no psuedonyms for these devices may be used. The "new_disk_name" may be up to 24 characters for $FLOPPY or 100 characters for $DISK, and may include any character except carriage return or semicolon. If a single quote is to be part of the new name, it must be immediately followed by a second single quote. The disk is renamed by first initializing the allocation maps, reading directly from the disk (via the floppy or hard disk driver) the map sector on which the disk ID is saved, altering it, and rewriting the sector. A second initialization is then made to update the disk name in memory.

I/O UNIT UTILIZATION

Unit 0: file I/O

EXAMPLES

RENAME $MYDOS/FILE.X FILE.Y

generates Assign and Rename requests for device MYDOS changing name of FILE.X to FILE.Y.

RENAME $FLOPPY:2 ID='MY NEWEST RIO DISK'

renames the diskette in drive 2.

SYNTAX

RESTORE_TABS file_name

DESCRIPTION

Replaces the current 134-character console tabbing environment with the tabs in the specified file. The file_name may be fully or partially qualified. The referenced file must have been previously created by the SAVE_TABS command.

I/O UNIT UTILIZATION

Unit 2: error messages
Unit 4: file I/O

EXAMPLES

RESTORE_TABS $MYDOS:TAB.ASM

replaces the current console tabbing environment with the tabs in the file TAB.ASM on device MYDOS.

SYNTAX

SAVE_TABS file_name

DESCRIPTION

Stores the current 134-character console tabbing environment into the specified file for possible later retrieval by the RESTORE_TABS command. The file_name may be fully or partially qualified. If the file all ready exists, it is deleted and recreated.

I/O UNIT UTILIZATION

Unit 2: error message
Unit 4: file I/O

EXAMPLE

SAVE_TABS LETTER.TABS

stores the current console tabbing environment into the file LETTER.TABS on the master device.

SYNTAX

```
SET  (CHRDEL=C | LINDEL=C | NULLCT=n | SPEED=NN | LFCNT = n
     ECHO ON | ECHO OFF | AUTOLF ON | AUTOLF OFF |
     PROPERTIES OF file_name TO plist |
     TYPE OF file_name TO type |
     SUBTYPE OF file_name TO subtype |
     ENTRY_POINT OF file_name TO nn |
     LOW_ADDRESS OF file_name TO |
     HIGH_ADDRESS OF file_name TO |
     STACK_SIZE OF file_name TO nn |
     BYTE_COUNT OF file_name TO nn |
     TABSIZE = n)*
```

DESCRIPTION

Sets a variety of system parameters. Any combination of the option list can be given in any order with each command entry.

CHRDEL=c

Sets the console driver single character delete symbol to character c. For example, typing 'SET CHRDEL=<control-H>' will cause all control-H's to be interpreted by the console input driver as a 'delete last character' command. (The characters '<' and '>' are not typed, but serve to illustrate that 'control-H' is a non-printing character.)

LINDEL=c

Sets the console driver line delete symbol to character c. For example, typing 'SET LINDEL=<rubout>' will cause the console driver to interpret <rubout> as a 'delete current line' command. (The characters '<' and '>' are not typed, but serve to illustrate that 'rubout' is a non-printing character.)

LFCNT = n

Sets to n the number of linefeed characters (0AH) the console driver automatically output after each carriage return if in AUTOLF=ON mode. Note that LFCNT=0 is equivalent to AUTOLF=OFF.

NULLCT=n

Sets the number of null characters (ASCII 0) to output by the system console driver after every carriage return to decimal value n. One null character is sufficient for CRT operation up to 19.2 Kbaud. Mechanical devices require longer head repositioning periods and thus a larger null count.

SPEED=nn

(MCZ only) Changes the serial communication port baud rate to the value given as nn. This port typically is used for terminal I/O. Any value from 20 baud to 4800 baud which is an even divisor of 4800, or 110, 9600, 19200, or 38400, can be selected.

TABSIZE=n

Redefines all tab settings to be every n columns, starting with the leftmost column as column 0. The default is every 8 columns.

PROPERTIES OF file_name TO plist

Sets the properties of 'file_name' to those given in the properties list plist. This list must be from W (write protect), E (erase protect), S (secret), L (locked), R (random), F (force memory allocation), or * (null, i.e., no properties). Locked files cannot have their properties altered. The properties of more than one filename may be set in one command by enclosing all the filenames within one set of parentheses.

SUBTYPE OF file_name TO subtype

Sets the file subtype of file 'file_name' to value 'subtype'. Only the least significant four bits of the value entered are used.

TYPE OF file_name TO type

Sets the file type of file 'file_name' to the type given - must be one of 'D' (directory), 'A' (ASCII), 'B' (binary) or 'P' (procedure).

ENTRY_POINT OF file_name TO nn

Sets the entry point field in the descriptor record of file_name to nn. This is the address to which control passes when the RIO Executive loads a procedure type file.

LOW_ADDRESS OF file_name TO nn

Sets to nn the lower boundary of the memory space which must be allocatable before a file_name can be loaded.

HIGH_ADDRESS OF file_name TO nn

Sets to nn the high boundary of the memory space which must be allocatable before file_name can be loaded.

STACK_SIZE OF file_name TO nn

Sets the size of the user stack which will be allocated before execution of file_name begins. Setting the stack size to zero will result in no stack allocation; the system stack will be used instead.

BYTE_COUNT OF file_name TO nn

Sets the 'bytes in last record' count for file_name to nn. This field is used by PLZ and BASIC to determine the number of valid data bytes in the last record of a file.

ECHO ON|OFF

Sets or resets the input character echo mode in CON (see section 3.4.3).

AUTOLF ON|OFF

Sets or resets the automatic line feed insertion mode flag in CON (see section 3.4.3).

I/O UNIT UTILIZATION

Unit 2: error messages
Unit 4: file I/O

EXAMPLES

```
SET LINDEL=! CHRDEL=@ NULLCT=2
```

Sets the line delete symbol to '!', the character delete symbol to '@', and the null count to 2.

```
SET PROPERTIES OF OS TO SWEL
```

Would give file 'OS' the properties secret, write protect, erase protect, and locked. Therefore it could never be altered or deleted without reformatting the disk.

```
SET PROPERTIES OF ($DFS:1/TEXT $ZDOS:0/TEXT
                   $DFS:0/STATUS.OBJ) TO *
```

Would clear the properties of 'TEXT' on DFS drive 1 of 'TEXT' on ZDOS drive 0 and of 'STATUS.OBJ' on DFS drive 0.

```
SET SUBTYPE OF $DFS/TEXT TO 8
```

Sets subtype of file TEXT on device MICRO.80 to 8H.

```
SET ECHO ON AUTOLF OFF
```

Sets the terminal mode to ECHO ON and AUTOLF OFF.

```
SET SPEED=9600
```

Sets the serial communication port baud rate to 9600.

```
SET ENTRY_POINT OF STAR_TREK TO 4419 BYTE_COUNT OF
STAR_TREK.S TO 38
```

Sets the entry point of STAR_TREK to 4419H, and the number of bytes in the last record of STAR_TREK.S to 38H.

APPENDIX A

RIO/ZDOS/DFS ERROR CODES

RIO

Completion Code	Meaning
40	Invalid Drive Name
41	Invalid or Inactive Device
42	Invalid Unit
43	Memory Protect Violation
44	Missing or Invalid Operand(s)
45	System Error
46	Illegal File Name
47	Non-existent Command
48	Illegal File Type
49	Program Abort
4A	Insufficient Memory
4B	Missing or Invalid File Properties
4C	I/O Error (IY->Vector)

ZDOS/DFS

Completion Code	Meaning
80	Operation Complete
81	Directory Format Error
82	Scratch File Created
83	File Name Truncated
84	Attribute List Truncated
C1	Invalid Operation (Request)
C2	Device Is Not Ready
C3	Write Protection
C4	Sector Address Error
C5	Seek Error
C6	Data Transfer Error
C7	File Not Found
C9	End of File Error
CA	Pointer Check Error
CB	File Not Open
CC	Unit Already Active (Open)
CD	Assign Buffer Full
CE	Invalid Drive Specification
CF	Logical Unit Table Full (>16 Open)

D0	Duplicate File
D1	Diskette ID Error
D2	Invalid Attributes
D3	Disk Is Full
D4	File Not Found in Proper Directory Record
D5	Beginning of File Error
D6	File Already Open on Other Unit
D7	Invalid Rename to Scratch File
D8	File Locked (Attempt to Change Attributes)
D9	Invalid Open Request
DA	Insufficient Memory for Allocation Maps

F0	Queue Full
F1	Unacceptable Request at this Time

SYNTAX

STATUS [0|1 ... 6|7]

DESCRIPTION

Lists statistics on how much of the disk on the specified drive has been used and how much of it remains available for new files. The default lists statistics on all drives which are ready.

Two error conditions are detected by the STATUS command. As part of the disk allocation maps kept on the disk, the number of free sectors and the number of allocated sectors are maintained. In the event they do not sum up to the number of sectors on the disk, the message

WARNING: DISK STATISTICS ARE INCONSISTENT

is printed. If the total number of sectors marked as unallocated in the sector map do not equal the free sector count, then the following message is printed:

WARNING: ALLOCATION IS INCONSISTENT

This is somewhat more serious than the previous error condition and could mean that sectors which are logically part of a file are marked unallocated in the allocation map. These errors may result from memory failure, disk write failure, deleting files with pointer errors, etc., and generally indicate reformatting of the diskette. However, it may still be possible to read all files from the disk and avoid loss of data.

I/O UNIT UTILIZATION

Unit 2: output listing

EXAMPLES

```
%STATUS 0

DRIVE 0   RIO MCZ SYSTEM DISK
659 SECTORS USED
1805 SECTORS AVAILABLE
%
```

VERBOSE
(Internal Command)

SYNTAX

Verbose

DESCRIPTION

Enter Verbose mode. Echo command strings as interpreted. Some commands test this mode before printing non-essential messages. See Brief command.

I/O UNIT UTILIZATION

None

5.42
XEQ

XEQ
(Internal Command)

SYNTAX

```
Xeq [* | nn [parameter_list]]
```

DESCRIPTION

Begin execution of last loaded command with optional parameter list, or begin execution at location nn with optional parameter list.

I/O UNIT UTILIZATION

None

EXAMPLES

```
XEQ
X *
```

Jumps to entry point of last loaded file.

```
X 5600 p1 p2
```

Jumps to address 5600H with INPTR referencing delimiter after '5600'.

```
X * p1 p2
```

Jumps to entry point of last loaded file with INPTR referencing delimiter after '*'.

5.43
EXPRESSION EVALUATION

EXPRESSION EVALUATION
(Internal Command)

SYNTAX

```
: expression
```

DESCRIPTION

Evaluates hex constant expressions left to right and prints result. Allowable operators are +, -, *, and /. Overflow is not detected.

EXAMPLES

```
: FD00-4400/80
0172

: 8732-4400/200
0021
```

CHAPTER 6

ZDOS

6.0 ZDOS OPERATION

This chapter covers the program interface for the ZDOS-II floppy disk file access system used under Zilog's RIO operating system. It describes the general interface structure and calling sequence, and, for each of the different requests, gives the details of the interface, a description of the actions taken, and a list and interpretation of the errors that could occur with that operation.

ZDOS-II is an improved version of ZDOS, the diskette access system which runs under earlier versions of Zilog software. For simplicity, in the remainder of this document ZDOS-II will be referred to as ZDOS.

ZDOS imposes a file structure on data stored on floppy disks. Data is stored as a sequence of records. All data records in a file are of the same length, and the length must be an integral number of sectors of the diskette media, and an integral power of two (valid record sizes are 128, 256, 512, 1024, 2048, and 4096 bytes). ZDOS maintains two pointers which are appended to each record. One is the disk address of the following record, the other is the disk address of the preceding record. The file is thus stored as a doubly-linked list of records.

Files are accessed by name through a directory. The directory is itself a file, and can be accessed as such by its name, 'DIRECTORY', which is the first entry in each directory. Unlike other files, however, it has a known first record so that it can be found; that is, it always begins at a fixed address known to the system. ZDOS 'DIRECTORY' files are type directory, subtype 0.

A scratch file is a slight exception to this. A scratch file is one which has existence only while it is active.

It is created by opening a file on a logical unit that has an assignment to a zero-length name (all assignments are initialized to scratch files, so opening a unit without making an assignment has the same effect). No directory entry is created, and the descriptor information is only stored internally rather than on the diskette. When the file is closed, any records which have been created on it are deallocated, and the file ceases to exist. Such a file is ideal for temporary storage of intermediate data. The programmer will find it advantageous to use scratch files whenever an application is suited to them, since opening them does not involve the directory operations that opening a named file does, and is therefore faster.

The directory is made up of sectors. Each sector is, in turn, made up of one or more variable length entries. Each entry consists of a single byte giving the length of the name, followed by the characters of the name, and a two byte pointer to the descriptor record (described below). The file names can be from 1 to 32 characters in length. The last entry in a sector is followed by a byte of -1 (0FFH). Directory entries do not span sector boundaries, so that if a new entry will not fit completely in a sector, it is put in the next one.

Occasionally, all the entries in a sector will be deleted. This happens relatively infrequently, and is indicated by the first byte of the sector (normally a length byte, which must be from 1 to 32) being the terminator byte, 0FFH.

The pointer contained in the directory entry for a file points to a special record, which is not one of the data records and is not included in the record count, called the file descriptor record. As its name indicates, it contains information describing the file to the system. Some of the information is also available to the user. Regardless of the length of the data records, the descriptor record is always 1 physical sector, or 128 bytes long. Of these, ZDOS has defined 40 bytes, leaving 88 which are available for programmer definition. Note, however, that there are some system conventions on how these remaining bytes will be used for some files. Most notably, procedure files contain segment addresses and lengths in this area (see Appendix J).

The information contained in the descriptor record is as follows:

Bytes 0-3	Reserved for future expansion
Bytes 4-5	File ID - currently unused
Bytes 6-7	Pointer to directory sector holding entry for this file
Bytes 8-9	Pointer to first data record of file
Bytes 10-11	Pointer to last data record of file
Byte 12	File type and subtype - see description with the OPEN request
Bytes 13-14	Record count
Bytes 15-16	Record length
Bytes 17-18	Block length - currently unused, and set to be same as record length
Byte 19	File properties - see description with the OPEN request
Bytes 20-21	Starting execution address for procedure files (entry point)
Bytes 22-23	Number of bytes in last record
Bytes 24-31	Date of creation
Bytes 32-39	Date of last modification
Bytes 40-127	Available for programmer definition

The Date of creation and Date of last modification are moved to the descriptor from the System Global Variable DATE at the appropriate times. Thus, if DATE is maintained to indicate the current date, then the descriptor record can give some historical information about the file.

The information stored in the descriptor (except for the first 12 bytes) is available to the program accessing the file at the time it is OPENed or while it is open by means of a QUERY ATTRIBUTES request. It can also be supplied at the time the file is created, or later by a SET ATTRIBUTES request, or when the file is UPDATED or CLOSED. See the appropriate request description for details.

Implicit in the description of accessing a file is the concept of the file pointer. There are actually three pointers. The one referred to as the pointer is the "current record pointer", the disk address of the record considered to be the current one. This is normally the record last handled, as, for example, in reading or writing. The "previous record pointer" contains the disk address of the record preceding the current one in the sequence of the file. The "next record pointer" contains the

disk address of the record following the current one in the sequence. A file is said to be active if these pointers are valid. OPENing a file consists of locating it in a directory, reading its descriptor record, and initializing these pointers.

A doubly-linked list provides some redundancy in establishing the sequence of the records. This redundancy is used when traversing the file to check the file integrity. For example, the forward pointer of the current record is used to establish the next record. When the next record is read, its back pointer is checked to make sure it indicates the current record. A failure of this or a similar check is what is referred to as a pointer error.

ZDOS is designed to operate with up to 8 floppy disk drives, each holding approximately 300 Kbytes. The standard MCZ has two drives, configured as drive 0 (also referred to as the system drive), and drive 2. The ZDS also has two drives configured as drive 0 (referred to as the system drive), and drive 1. When a file is to be located, and the drive is not specifically indicated (equivalent to specifying '*'), the drives are searched in order, starting with drive 1, and continuing through the highest disk which is attached and ready, and finally, if still unsuccessful, concluding with drive 0.

Similarly, if a file is to be created without specifying which disk it is to be on, it will be created on the first disk which is attached and ready in the same search order.

Under RIO, all I/O calls pass through the operating system where they are routed to the required device driver according to the logical unit being requested and the current routing for that unit. Calling parameters are passed to the drivers via a 13-byte "parameter vector" which is pointed to by the IY register. There are two ways in which I/O calls are handled by the drivers. The driver may perform the entire operation, then return to the calling program (referred to as "return on completion"), or it may perform only the initial setup necessary, then return to the calling program and let the operation proceed to completion under interrupt control (referred to as "immediate return"). A completion code is provided in the parameter vector to indicate when the operation is complete and signal any unusual circumstances of the completion.

The parameter is set up as follows:

(IY)	Logical unit number (1 byte) - identifies the particular dataset being accessed. Used by OS to route the call to the correct driver.
(IY+1)	Request code (1 byte) - identifies the action to be taken. Two request codes are given for each operation in the following list. One is even, the other odd. If the first is used, the return will be on completion of the operation. If the second is used, then return will be immediate, with completion occurring under interrupt control.
(IY+2)	Data transfer area (2 bytes) - gives the address at which data transfer is to begin. If no data transfer is expected for a given request, this field should be zero.
(IY+4)	Length (2 bytes) - gives the length of the operation. For most operations, this is the number of bytes to transfer, but refer to specific operation descriptions. If no data transfer is expected, the length should be zero. On return, this gives the length actually completed.
(IY+6)	Completion return address (2 bytes) - specifies address to branch to when the operation completes if the request code specified immediate return. If return on completion is specified, this element is ignored.
(IY+8)	Error return address (2 bytes) - if non-zero, specifies the address to branch to if an error occurs. The error will still be indicated in the completion return code. If the address is zero, return is as though there were no error.

(IY+10) Completion code (1 byte) - indicates when the operation is complete. This byte is set to zero when the call is made. Bit 7 is set when the operation is complete. Bit 6 is set if an error occurred. The remainder of the byte will contain a code indicating the nature of the difficulty or error. A normal completion will thus contain a code of '80'.

(IY+11) Supplemental parameter information - some requests require special information which does not fit into the general structure of the parameter vector. This information is supplied by the supplemental parameter vector. If two bytes or less (e.g., a disk address), it is normally put here. Otherwise an address pointer to an area containing the information is placed here.

Following is a list of error codes with their meaning. They will be discussed in detail under the description of each operation with which they can occur.

CODE (Base 16)	MEANING
C1	Invalid operation
C2	Not ready
C3	Protection
C4	Sector error
C5	Seek error
C6	Data transfer area
C7	File not found
C8	--
C9	End of file error
CA	Pointer check error
CB	File not open
CC	Unit already active
CD	Assign buffer full
CE	Invalid disk drive
CF	Logical unit table full
D0	Duplicate file
D1	Diskette ID error
D2	Invalid attributes
D3	Disk full
D4	File not in proper directory record
D5	Beginning of file error
D6	File already open (on another unit)
D7	Invalid rename
D8	File locked (attempt to change attributes)
D9	Invalid open request
DA	Insufficient memory for allocation maps

The following are warning codes. They do not have bit 6 (the error bit) set, and do not cause transfer to the error return address.

81	Directory format error
82	Scratch file created
83	File name truncated
84	Attribute list truncated

There are some errors which are either not associated with a particular request, or can occur on almost every request, that are described in detail here. An INVALID OPERATION error (code C1) will occur anytime the request code is not one of the valid operations for the device addressed. ZDOS will respond with this code to operation READ LINE (0C), WRITE LINE (10), and WRITE DIRECT (14), as well as anything 32 or over.

Disk I/O errors can occur on almost any operation. There are 5 different errors that can occur, though some of them have other meanings as well. A NOT READY error (code C2) indicates that an attempt was made to access a drive which was not asserting its READY signal. This may also signify designation of an operation to a drive which was recorded as being not ready at initialization. A WRITE PROTECT error (code C3) indicates that an attempt was made to write on a disk which is physically write protected. This could also indicate a request which would cause a change in a file which is (software) write protected, or a request which would remove records from a file which is erase protected. A SECTOR error (code C4) is always a media or hardware problem, indicating that the sector header information read did not agree with the location recorded on the disk. A TRACK ERROR (code C5) indicates that there was a hard seek error, or that the sector address header was destroyed, or else that an invalid track was requested from the floppy driver (a ZDOS software error!). A CRC ERROR (code C6) indicates that there was a data error in transmitting from the disk to memory, or that the data was written incorrectly on the disk in the first place.

Another error which could conceivably occur almost anytime is LOGICAL UNIT TABLE FULL (code CF). ZDOS maintains an internal mapping between the 255 possible logical unit designations and the 16 for which it has space. The first reference to a new unit causes it to be entered in this map. If the unit given is not found in the map, and there are no empty entries in it, then this error is returned. An entry is removed from this map when a file is closed, or when one is found to be not open when it should be. The table is also cleared when an INITIALIZE request is done.

Request vector:

Logical unit - ignored
Request code - 00 or 01
Data transfer area - ignored
Length - ignored. Zero will be returned.
Completion return address
Error return address
Completion code
Supplemental parameter information - none

Action:

All 16 logical units are flagged as not open. The logical unit map is cleared. Memory is allocated at the top of the available space for each map, and the maps are read in from the disk. A flag word is constructed indicating which drives are ready.

Possible errors:

Disk SECTOR, SEEK, or DATA TRANSFER errors (C4, C5, or C6). The initialization is not completed if one of these occur.

INSUFFICIENT MEMORY (code DA) - There was insufficient memory available to fulfill one or more of the requests for space for the allocation maps.

Request vector:

Logical unit
Request code - 02 or 03
Data transfer area - ignored
Length - ignored. Zero will be returned.
Completion return address
Error return address
Completion code
Supplemental parameter information - pointer to area containing the following:
1st byte of area - ignored by ZDOS. (This byte does control system functions however. See Section 3.3.)
2nd byte of area - character designating physical disk drive, either '0' through '7', or '*'.
3rd byte of area - length in bytes of file name. A 0 length name indicates a scratch file. The maximum length name is 32 characters.
4th and following bytes - the filename.

Action:

The filename given is associated with (assigned to) the given logical unit for subsequent I/O operations. Any previous assignment to the same logical unit is nullified. The filename is stored in a buffer for later use when the logical unit is opened. When the file is opened, the filename is removed from the buffer. Thus, if a file is opened and closed, there must be a new assignment before it can be reopened.

Possible errors:

FILE ALREADY OPEN (code CC) - an attempt to assign a file to a unit which is currently active. The assignment is not made.

INVALID DRIVE (code CE) - some drive is specified other than '*' or '0'-'7'.

ASSIGN BUFFER FULL (code CD) - the buffer used for storing filenames after assignment prior to files being opened is too full to hold the name assigned. The assignment is not made.

NAME TOO LONG (code 83) - this circumstance is only a warning. The name given was indicated as being longer than the maximum length and was truncated accordingly.

Request vector:

Logical unit
Request code - 04 or 05
Data transfer area - A pointer to an area containing the attributes that the file is to be created with, if it gets created, or where the attributes can be returned if the file exists. See below for a detailed description of these attributes. If this pointer is zero, a set of default attributes will be supplied if the file is created. Nothing will be returned if the file is found.
Length - The number of bytes to transfer to/from the attributes in the data transfer area. If this is zero, or less than the minimum set of attributes for a new file, and the file needs to be created, the balance will be supplied from defaults. See below for a detailed description of attributes.
Completion return address
Error return address
Completion code
Supplemental parameter information - Pointer to an area containing the following:
1st byte of area - designation of the type of action to be performed on the open. See below for possibilities.
2nd byte of area - A byte for returning a character representing the disk drive the file was opened on. A character '0'-'7' will be returned in this byte. If a new assignment is to be made, such a character or '*' should be supplied by the calling program to indicate where the file is to be searched.

3rd byte of area - length of file name. The open request can do its own assignment, removing the necessity for two calls to ZDOS. If this is done, it will override any assignment done previously. This supplemental information should look exactly like the supplemental information for an assign request. If no assignment is to be made, however, this byte should be -1 (OFFH).

4th byte and following - filename if there is one.

File attributes:

Each file has a 128-byte record referred to as the file descriptor record. This contains information concerning the type of file, where it is on the disk, how it is organized, etc. Only 40 of the 128 bytes are used by the system, leaving the remainder for possible use by the user. When a file is created, the organization, etc., must be specified by the user, either explicitly or by default. Similarly, when a file is opened, the information about the organization may be needed by the program. In order to accomplish both these ends, that portion of the file descriptor record which may be of use to the programmer can be passed back and forth. It is laid out as follows:

1 Type and subtype. There are 4 types of files recognized by the system. Each is assigned one of the top four bits of this word. The bottom four bits are available for user defined subtypes.
Bit 7 - procedure type files.
Bit 6 - Directory files.
Bit 5 - ASCII files.
Bit 4 - Data files.

The default is ASCII subtype 0 (20H).

2-3 Record count. Number of records in the file.

4-5 Record length in bytes. The default is 128. If zero is specified, the default will be assumed.

6-7 Block length in bytes. This has to do with logical blocking of records, which is currently unimplemented. This is therefore set to the record length.

8 File properties. The following bits are assigned:
Bit 7 - Write protection - the file cannot be changed.
Bit 6 - Erase protection - nothing can be removed from the file. Write protection implies erase protection, but not vice versa.
Bit 5 - Locked - No attributes can be changed.
Bit 4 - Secret - the file will not appear in normal directory listings.
Bit 3 - Random - file is set up for random access. This has not been defined at this time.
Bit 2 - FORCE file loading.
Bit 1 - Reserved for system use.
Bit 0 - Reserved for system use.

9-10 Start address - Address at which execution of a procedure file should begin. Not used by ZDOS itself.

11-12 Reserved for system use.

13-20 Date of creation. This is supplied by ZDOS from the system global DATE.

21-28 Date last written. This is supplied by ZDOS from the system global DATE.

29-116 Available for user definition.

Types of open requests:

There are several ways the activation of a file may be handled. These are specified by the 1st word of the supplemental parameter information. A file may be opened for either random or sequential access. Random access is specified by setting bit 3 of this word. Currently, a file being open for random access has two implications. One is that the READ DIRECT request will be accepted (it will be refused with an INVALID REQUEST error otherwise). The other is that in each record oriented operation, the disk address of the first record involved will be returned to the calling program in the supplemental parameter information field of the parameter vector.

There are five mutually exclusive ways that the cases of file not found/ file found may be handled. These are specified in the bottom 3 bits of this word. They are as follows:

Open for input - 0 - if the file exists, it will be activated with the pointer ahead of the first record. If it does not exist, a FILE NOT FOUND error (code C7) is returned.

Open for output - 1 - If the file exists, it is activated and all its records are deleted. If it does not exist, it is created.

Open new file - 2 - (also referred to as open for nondestructive output) - if the file exists, a DUPLICATE FILE error (code D0) is returned, and the file is not activated. If the file does not exist, it is created.

Open for append - 3 - if the file exists, it is activated with the pointer positioned at the last record. If it does not exist, it is created.

Open for update - 4 - if the file exists, it is activated with the pointer ahead of the first record of the file. If it does not exist, it is created.

Action:

If a filename assignment is specified (3rd byte of the supplemental information is not -1), the ASSIGN subroutine is called as though an ASSIGN request had been made. If there is a filename assigned to the unit, i.e., if it is not assigned to a scratch file, the directory on the specified drive is searched for the filename. If the drive is specified as '*', a check is made to determine the ready status of all drives, then each ready drive is searched, from drive 1 to drive 7, followed by drive 0, until the file is found or all drives have been searched.
The ID of the diskette which holds or will hold the file is read into a buffer and compared against the ID on the corresponding map in memory. If they do not match, and no other unit has a file open on the same physical drive, a new map (and ID) will be read into memory. If another unit is open on that drive, a WRONG DISKETTE error will be returned and the file will not be activated. If the file is found, its descriptor record is read, the relevant parts are moved into the active file table entry for the unit, and, if requested, moved to the user's data transfer area. The file is then flagged as open. If the file is to be created, the descriptor record is created in a buffer, then moved to the active file table, and, if requested, to the user's data area. If the file is not a scratch (no-name) file, the descriptor record is written out to the disk and a directory entry is created.

Possible errors:

All DISK ERRORS are possible. NOT READY (code C2) may indicate designation of a specific drive that was recorded as 'not ready' the last time the ready status was checked.

PROTECTION (code C3) - may occur as a disk error if the diskette is write-protected, or may occur by an attempt to open an existing file which is write- or erase-protected for output (thus deleting its records). In that case, the file is opened but its records are not deleted.

UNIT ALREADY OPEN (code CC) - the logical unit is already active. It must be closed, or an initialize operation must be performed, before it can be OPENed again. No action is taken.

WRONG DISKETTE (code D1) - the disk ID of the diskette in the drive does not match the ID in memory. Usually indicates that the disks have been switched since an INITIALIZE operation was performed, or that a program has overwritten the maps in memory. The file is not opened.

FILE NOT FOUND (code C7) - the open request was for input, and the file designated does not exist.

POINTER ERROR (code CA) - could occur if the pointers linking the segments of the directory together have been destroyed or overwritten, or if the file exists and the pointers for the descriptor record are incorrect, or, in deleting the records of an existing file, a pointer mismatch occurs.

DUPLICATE FILE (code D0) - request to open a new file when the file already exists. The file is not activated.

INVALID ATTRIBUTE (code D2) - one of the attributes specified for the creation of the file was invalid. This may be that more than one (or none) of the four mutually exclusive types was specified, or that an invalid record size was specified. The file is activated with the defaults substituted for the erroneous attributes.

DISK FULL (code D3) - there was no space to allocate a descriptor record, or a new directory record if one needed to be allocated. Can only occur if the file is being created.

FILE ALREADY OPEN (code D6) - the file requested to be opened on this unit is already active on another unit. The unit is not activated.

PROPERTIES PROTECTION (code D8) - an attempt to change attributes on a locked file. The attributes are not changed.

INVALID OPEN REQUEST (code D9) - a type of open request which was not input, output, newfile, append, or update was specified. No action is taken.

INSUFFICIENT MEMORY (code DA) - if additional disks have been inserted prior to the open request and insufficient memory is available for additional allocation maps, this error will be returned.

The following are warning codes, and will not cause the error return branch to be taken.

DIRECTORY FORMAT ERROR (code 81) - Indicates the format of one or more directory records is erroneous. The record can still be read, but its data is suspect.

SCRATCH FILE (code 82) - informative message that a scratch file has been created.

ATTRIBUTES TOO LONG (code 84) - more than 116 bytes of attribute information were requested. Only 116 bytes were transferred.

In addition, if an assign is implicit in the open request, any error that can occur with assign could occur.

Request vector:

Logical unit
Request code - 06 or 07
Data transfer area - a pointer to an area which may contain attributes to replace those of the file. The format is the same as for the OPEN request. If no replacement of attributes is desired, the data transfer area should be zero.
Length - Number of bytes to move to the descriptor record from the data transfer area. If no data is to be moved, it should be zero. The maximum is 116 decimal bytes.
Completion return address
Error return address
Completion code
Supplemental parameter information - none

Action:

If the logical unit is a scratch file, the file is erased. If there have been any changes in allocation to the file, or if new attributes are to be written, its descriptor record is read, updated, including moving in any new attributes supplied by the calling program, and rewritten, and the allocation map is rewritten. The file is then flagged as being closed, and indicators are set to indicate an assignment to a scratch file.

Possible errors:

FILE NOT OPEN (code CB) - the logical unit is not active. No action is taken.

WRONG DISKETTE (code D1) - the diskette ID on the disk does not agree with the ID in memory. Usually indicates disks have been changed since an INITIALIZE was performed, or that a program wrote over the allocation maps. The file is not closed. Either the correct disk must be re-inserted, or an INITIALIZE must be performed, which will result in the file status information being cleared.

INVALID ATTRIBUTE (code D2) - if attributes are specified for replacement and the type is invalid, this error will result. The type is left unchanged, but everything else is done.

All DISK ERRORS are possible.

POINTER ERROR (code CA) - indicates a pointer mismatch occurred while deleting the records of a scratch file, or that the pointers on the descriptor record were incorrect. If it occurs while reading the descriptor record, the file is deactivated but the descriptor is not updated.

The following warnings are possible:

ATTRIBUTES TOO LONG (code 84) - more than 116 bytes of replacement attributes were specified. Only 116 bytes were transferred.

Request vector:

Logical unit
Request code - 08 or 09
Data transfer area - ignored
Length - ignored
Completion return address
Error return address
Completion code
Supplemental parameter information - none

Action:

The file pointer is positioned at the descriptor record, with the first record as the next record. This is the position the file pointer assumes when the file is opened for other than append, or when it is created. If there are no records in the file, the next record pointer is null.

Possible errors:

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

Request vector:

Logical unit
Request code - 0A or 0B
Data transfer area - the address to which data should be transferred
Length - the number of bytes to transfer. If this number is not an integral multiple of the record size, it will be rounded up until it is. On return, this will contain the actual number of bytes transferred.
Completion return address
Error return address
Completion code
Supplemental parameter information - if the file is open for random I/O, this field should contain the address of a three-byte area where the disk address of the first record read will be returned. Otherwise, it is unused.

Action:

Data is read from the file, starting at the next record, into the data transfer area. The pointer is left on the last record read. If the file is open for random I/O, the disk address of the first record read is returned in the field pointed to by the supplemental parameter information. The third byte of this address will always be zero.

Possible errors:

All DISK ERRORS except PROTECTION (code C3) are possible.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

END OF FILE (code C9) - the last record of the file was read and the given length had not yet been fulfilled. The length returned reflects the number of bytes actually read.

POINTER ERROR (code CA) - a pointer mismatch occurred. The reading stops at the point it is detected. The length returned will include the record which had the error.

Request vector:

Logical unit
Request code - 0E or 0F
Data transfer area - the address from which data is to be transferred.
Length - the number of bytes of data to transfer. If this number is not an integral multiple of the record size, it will be rounded up until it is. On return, this will contain the actual number of bytes transferred.
Completion return address
Error return address
Completion code
Supplemental parameter information - if the file is open for random I/O, this field should contain the address of a three-byte area where the disk address of the first record written will be returned. If not, it is unused.

Action:

New records are created and filled with data from the data transfer area. The new records are inserted after the current one. The pointer is left at the last record written. The next record pointer remains on the same record it was prior to the operation.

Possible errors:

All DISK ERRORs are possible except CRC (code C6). PROTECTION (code C3) will also be returned by ZDOS if the file is write protected.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

DISK FULL (code D3) - there is no room on the disk to allocate a new record. Records which will fit are written. The length returned reflects the number of bytes written before the disk filled up.

Request vector:

Logical unit
Request code - 12 or 13
Data transfer area - the address from which data is to be transferred.
Length - if the length is zero, no data will be transferred. Otherwise, one record will be transferred. On return, length will contain the number of bytes transferred.
Completion return address
Error return address
Completion code
Supplemental parameter information - if the unit is open for random I/O, this field should contain the address of a three-byte area where the disk address of the record will be returned. Otherwise, it is unused.

Action:

Data is moved from memory to the file, replacing the data in the current record. No new records are created, and the record pointer is not moved.

Possible errors:

All DISK ERRORs except CRC (code C6) are possible. PROTECTION (code C3) will also be returned by ZDOS if the file is write protected.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

Request vector:

Logical unit
Request code - 16 or 17
Data transfer area - ignored.
Length - the number of bytes of data to be removed from the file. If this number is not an integral number of records, it will be rounded up to the next full record. On return, length will contain the number of bytes deleted.
Completion return address
Error return address
Completion code
Supplemental parameter information - none.

Action:

Starting at the current record, records are removed from the file, and the space taken up by them deallocated (made available), until the given number of bytes have been removed. The current record pointer is left on the record preceding those deleted. The next record pointer is left at the record following those deleted.
If the file is currently positioned on the descriptor record (top of the file), the pointer will be advanced to the first record before the operation is started. This is not counted as one of the records deleted. After the operation, the pointer will again be on the descriptor record.

Possible errors:

All DISK ERRORs are possible. PROTECTION (code C3) will also be returned by ZDOS if the file is either write or erase protected.

END OF FILE (code C9) - the last record of the file was deleted, and the length specified had not yet been exhausted. The number of bytes returned will indicate the number deleted, including the last record. The pointer is left at the record preceding the first one deleted, with the next record pointer being null.

POINTER ERROR (code CA) - a pointer mismatch occurred while traversing the records of the file in deleting them. Records are deleted up to the one preceding the mismatch.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

Request vector:

Logical unit
Request code - 18 or 19
Data transfer area - ignored
Length - ignored. The total number of bytes deleted is returned.
Completion return address
Error return address
Completion code
Supplemental parameter information - none

Action:

All records from the current one to the end of the file are removed. The pointer is left on the record preceding those deleted. The next record pointer is null. If the pointer is on the descriptor before the operation, it is moved forward before deletion begins.

Possible errors:

All DISK ERRORS are possible. PROTECTION (code C3) will also be returned if the file is write- or erase-protected.

POINTER ERROR (code CA) - will occur if, in traversing the records of the file, a pointer mismatch is found. The records will be deleted up to, but not including, the record preceding the mismatch.

FILE NOT OPEN (code CB) - the requested logical unit is not active. No action is taken.

Request vector:

Logical unit
Request code - 1A or 1B
Data transfer area - ignored
Length - ignored. The total number of bytes deleted is returned.
Completion return address
Error return address
Completion code
Supplemental parameter information - none

Action:

All records of the file are deallocated (their space is made available), the descriptor record is deallocated, and the directory entry for the file is removed from the directory, thus rendering the file completely inaccessible. The file does not have to be open, but it must have been assigned.

Possible errors:

All DISK ERRORS are possible. In addition, PROTECTION (code C3) will be returned by ZDOS if the file is write- or erase-protected. NOT READY (code C2) will be returned if the specified drive for the file is not loaded and ready.

FILE NOT FOUND (code C7) - The named file cannot be located on the drive specified. Will not occur if the file is open when the request is given.

POINTER ERROR (code CA) - A pointer mismatch occurred either while locating the file or while traversing the records of the file. All records beyond the pointer mismatch will remain allocated and thus be unavailable for further use.

INVALID DRIVE (code CE) - The drive specified in the assignment was something other than '0' - '7'. '*' is not a valid specification for a file being erased.

WRONG DISKETTE (code D1) - the ID for the diskette the file is on does not match the ID in memory for that diskette. Usually indicates that the diskette has been changed or that a program has overwritten the ZDOS map area. The file is not erased.

FILE NOT FOUND IN DIRECTORY (code D4) - this error indicates that no directory entry for the file could be found in the segment of the directory indicated by the descriptor record. The records will be deallocated, but a directory entry may remain somewhere else. If this error occurs, it is best to copy all remaining files to another diskette and reformat the one in question, as any further access to the bad file is liable to cause pointer errors and other complications.

FILE ALREADY OPEN (ON ANOTHER UNIT) (code D6) - this error will result from an attempt to erase on one unit a file which is currently active on another unit. No action will be taken.

Request vector:

Logical unit
Request code - 1C or 1D
Data transfer area - address to which data will be read
Length - number of bytes of data to read. If this is not equal to an integral number of records, it will be rounded up until it is. On return, length will contain the number of bytes actually transferred.
Completion return address
Error return address
Completion code
Supplemental parameter information - if the file is open for random I/O, this field should contain the address of a three-byte area where the disk address of the first record read will be returned.

Action:

This request is useful when a block of data is to be read, modified, and rewritten. Starting at the record following the current one, data is transferred from the file to memory and simultaneously removed from file (that is, the records are deallocated). The current record at the start of the operation will remain the current record when it is complete, thus leaving the pointer in position to write data back in the same position from which it was just read. The record after the last one read will be indicated as the next record.

Possible errors:

All DISK ERRORS are possible. In addition, PROTECTION (code C3) will be returned by ZDOS if the file is write- or erase-protected.

END OF FILE (code C8) - the last record of the file was read without exhausting the length specification. The number of bytes read is returned in the length field, and the next record pointer is null.

POINTER ERROR (code CA) - a pointer mismatch occurred in going from one record to the next. The data transfer stops with the record in error. The length field indicates how many bytes were transferred prior to the error.

FILE NOT OPEN (code CB) - the logical unit being requested is not active. No action is taken.

Request vector:

Logical unit
Request code - 1E or 1F
Data transfer area - address to which data should be read
Length - number of bytes to read. If this is zero, no data will be transferred. Otherwise, one record of data will be transferred. The number of bytes actually transferred will be indicated here on return.
Completion return address
Error return address
Completion code
Supplemental parameter information - if the file is open for random I/O, this field should contain the address of a three-byte area where the address of the current record will be returned. Otherwise, it is unused.

Action:

Unless the length specification is zero, one record's worth of data is transferred from the current record. The pointer is left unmoved.

Possible errors:

All DISK ERRORS except PROTECTION (code C3) are possible.

POINTER ERROR (code CA) - if the back pointer of the current record does not indicate the previous record, a pointer error is reported. The data is still transferred.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

BEGINNING OF FILE (code D5) - the pointer is on the descriptor record, which cannot be read. No data is transferred, and a null address is returned if open for random I/O.

Request vector:

Logical unit
Request code - 20 or 21
Data transfer area - address to which data should be read
Length - number of bytes to read. If this is zero, no data will be transferred. Otherwise, one record of data will be transferred. The number of bytes actually transferred will be indicated here on return.
Completion return address
Error return address
Completion code
Supplemental parameter information - if the file is open for random I/O, this field should contain the address of a three-byte area where the disk address of the previous record will be returned. Otherwise, it is unused.

Action:

Unless the length is zero, the record preceding the current one is read. In either case, the pointer is backed up one record. The current record will become the next record, the previous one the current record, and the one preceding the previous record will become the new previous record.

Possible errors:

ALL DISK ERRORS except PROTECTION (code C3) are possible.

POINTER ERROR (code CA) - if the forward pointer on the previous record does not indicate the current one, a pointer error is reported. The data is still transferred.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

BEGINNING OF FILE (code D5) - if the record indicated is the descriptor record of the file, the pointer is positioned at the beginning and a previous record is meaningless. No action is taken. A null disk address is returned if the file is open for random I/O.

Request vector:

Logical unit
Request code - 22 or 23
Data transfer area - memory address to which data is to be transferred.
Length - number of bytes of data to transfer. If this number is not an integral number of records, it will be rounded up until it is. The number of bytes actually transferred will be reported in this field.
Completion return address
Error return address
Completion code
Supplemental parameter information - a pointer to a three-byte area containing the disk address of the first record to be read. The third byte of this area should be zero. It is the calling program's responsibility to be certain this sector is a part of the file being accessed. This field will be unchanged on return.

Action:

The record whose disk address is given is read as though it were the next record, but no pointer checking is done on it. If more than one record is specified for the length, subsequent records are read just as in the read binary request. This request is only valid if the file has been opened for random I/O.

Possible errors:

INVALID REQUEST (code C1) - if the file is not open for random I/O, the read direct request will be rejected with this error.

All the errors of the READ BINARY request apply for the READ DIRECT request as well.

Request vector:

Logical unit
Request code - 24 or 25
Data transfer area - ignored
Length - the number of records (not bytes) to be skipped. On return, the number of records skipped will be reported.
Completion return address
Error return address
Completion code
Supplemental parameter information - if file is open for random I/O, this field should contain the address of a three-byte area where the disk address of the first record skipped will be returned. Otherwise, none.

Action:

The current record pointer is advanced through the file by the number of records indicated. No data is transferred.

Possible errors:

All DISK ERRORs except PROTECTION (code C3) are possible.

END OF FILE (code C9) - the last record of the file was reached while there remained records to skip. The pointer is left at the last record of the file, with a null pointer for the next record.

POINTER ERROR (code CA) - a pointer mismatch occurred while traversing the records of the file. The current record pointer is left at the record preceding the mismatch.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

Request vector:

Logical unit
Request code - 26 or 27
Data transfer area - ignored
Length - the number of records (not bytes) to be skipped. On return, the number of records skipped will be reported.
Completion return address
Error return address
Completion code
Supplemental parameter information - if the file is open for random I/O, this field should contain the address of a three-byte area where the disk address of the record preceding the current one will be reported. Otherwise, none.

Action:

The current record pointer is moved backward through the file by the number of records indicated. No data is transferred.

Possible errors:

All DISK ERRORs except PROTECTION (code C3) are possible.

POINTER ERROR (code CA) - a pointer mismatch occurred while traversing the records of the file. The current record pointer is left indicating the record following the error.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

BEGINNING OF FILE (code D5) - the beginning of the file was reached without exhausting the given record count. The pointer is left positioned on the descriptor, with the first record of the file being next.

Request vector:

Logical unit
Request code - 28 or 29
Data transfer area - ignored
Length - ignored. Zero will be returned.
Completion return address
Error return address
Completion code
Supplemental parameter information - if the file is open for random I/O, this field should contain the address of a three-byte area where the disk address of the last record in the file will be returned. Otherwise, it is unused.

Action:

The file is positioned with the current pointer indicating the last record of the file.

Possible errors:

All DISK ERRORs are possible with the exception of PROTECTION (code C3).

POINTER ERROR (code CA) - the forward pointer of the last record was not null, indicating it was not, in fact, the last record.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

Request vector:

Logical unit
Request code - 2A or 2B
Data transfer area - ignored
Length - ignored
Completion return address
Error return address
Completion code
Supplemental parameter information - a pointer to an area configured as follows
1st byte - length of name
2nd and following bytes - new name

Action:

The file on the unit requested is given the new name which is contained in the supplemental parameter vector. The file may be open. If it is not, there must be a pending assignment for it, so that it can be opened. If it is not already open, it will be activated, then its directory entry removed from the directory and a new one created. An open scratch file may be recovered (requiring its descriptor record to be created), but a named file cannot be renamed to a scratch file. Finally, if the file was not open at the start of the operation, it is deactivated.

Possible errors:

All DISK ERRORs are possible.

FILE NOT FOUND (code C7) - the unit being accessed was not open and the file assigned to it does not exist. Will also occur if the unit is assigned to a scratch file which has not yet been created. No action is taken.

POINTER ERROR (code CA) - the back pointer from the descriptor record did not point to the directory. No further action is taken.

DUPLICATE FILE (code DO) - a file of the same name as the new name already exists. The file is not renamed.

DISK FULL (code D3) - it was necessary to create a new directory record to contain the new name, and the disk was too full to allow it, or too full to allow creation of the descriptor record if renaming a scratch file.

FILE NOT IN PROPER DIRECTORY RECORD (code D4) - no directory entry for the file exists in the directory record indicated by its descriptor. No action is taken.

FILE ALREADY OPEN ON ANOTHER UNIT (code D6) - the file assigned to the unit being accessed is currently active on another logical unit. No action is taken.

INVALID RENAME (code D7) - attempt to rename a file either to a scratch file (zero length name) or to a name longer than the maximum name length (32 characters). The file is not renamed.

Request vector:

Logical unit
Request code - 2C or 2D
Data transfer area - address of attributes to be assigned to the file. Format is described under the OPEN request.
Length - number of bytes of attribute information to be used.
Completion return address
Error return address
Completion code
Supplemental parameter information - none

Action:

If there have been any changes to the file, or if there are attributes to be updated, the descriptor is read, updated, and rewritten, and the allocation map is rewritten. The attributes of the file can be changed with the update request in the same way as by the close request. The file remains active.

Possible errors:

All DISK ERRORs are possible.

POINTER ERROR (code CA) - will occur if the back pointer for the descriptor does not indicate the directory record for the file.

FILE NOT FOUND (code CB) - the logical unit being accessed is not active. No action is taken.

WRONG DISKETTE (code D1) - the diskette ID for the drive for this file does not agree with the ID in memory for that drive. Usually indicates the diskette has been changed since the file was opened, or that a program wrote across the allocation maps. No action is taken.

INVALID ATTRIBUTES (code D2) - one or more of the attributes being supplied either is or was invalid. The attributes checked are type (there should be exactly one of the most significant bits on) and record length. The attribute which was wrong is left as it was, and the remainder of the process is carried on.

PROPERTY PROTECTION (code D8) - if the file is locked, no attributes are to be changed. An attempt to do so results in this error.

Request vector:

Logical unit
Request code - 2E or 2F
Data transfer area - address of attributes to be assigned to the file. Format is described under the OPEN request.
Length - number of bytes of attribute information to be used.
Completion return address
Error return address
Completion code
Supplemental parameter information - none

Action:

The descriptor record is read and updated from the current information and the attributes supplied and rewritten.

Possible errors:

All DISK ERRORS are possible.

POINTER ERROR (code CA) - will occur if the back pointer for the descriptor does not indicate the directory record for the file.

FILE NOT FOUND (code CB) - the logical unit being accessed is not active. No action is taken.

INVALID ATTRIBUTES (code D2) - one or more of the attributes being supplied either is or was invalid. The attributes checked are type (there should be exactly one of the most significant bits on) and record length. The attribute which was wrong is left as it was, and the remainder of the process is carried on.

PROPERTY PROTECTION CODE (code D8) - if the file is locked, no attributes are to be changed. An attempt to do so results in this error.

Request vector:

Logical unit
Request code - 30 or 31
Data transfer area - address for return of attributes of the file.
Length - number of bytes of attribute information to be returned.
Completion return address
Error return address
Completion code
Supplemental parameter information - none

Action:

This command is used for obtaining attribute information, such as the record count, while a file is open. The descriptor record is read, the current information merged into it, and as many bytes of attribute information as have been requested are returned to the user. See the OPEN request for the format of the returned bytes.

Possible errors:

All DISK ERRORS except PROTECTION (code C3) are possible.

POINTER ERROR (code CA) - the back pointer of the descriptor record does not indicate the directory record. No data is returned.

FILE NOT OPEN (code CB) - the logical unit being accessed is not active. No action is taken.

ATTRIBUTE LIST TRUNCATED (code 84) - this code is a warning and will not cause an error branch. It indicates that more than 116 bytes of attribute information were requested. Only 116 bytes were transferred.

CHAPTER 7

DFS

7.1 ZILOG DISK CONTROLLER

The interface from a Zilog Z-80 System to high-speed disk units consists of an intelligent disk controller in a module separate from the system, and a high speed serial interface link from the system to the controller. The controller contains its own Z-80 CPU, sockets for 8K of ROM (of which only 3K is currently used), 16K of static RAM, Direct Memory Access (DMA) control circuitry, the data decode/encode interface circuitry and miscellaneous control signal interface logic. The interface is to Caelus Model 206R or similar units. It will support up to four disk drives with appropriate daisy-chained cabling.

The software for this configuration makes the disk appear as another file system, functionally identical to the ZDOS file system which exists for the floppy disks.

The Disk File System (DFS) software is capable of running in two versions. On the MCZ 1/20, the system is bootstrapped from the floppy disks as a standard system. The system resident portion of DFS must be loaded from ZDOS by the command ACTIVATE $DFS. ZDOS is at 2A00, immediately above the console driver for RIO, and DFS is placed at the top of memory. DFS contains an entry point for DISK, the low level disk controller used by some utilities; it is 2 greater than the entry point for DFS, and should be activated with an address at the same time DFS is activated, thus:

```
ACTIVATE $DFS;X * $DISK E002
```

If DFS is to be run as the master device, then the command MASTER $DFS should follow these. At this point ZDOS can be deactivated, thus effecting a large savings in memory space, unless it is needed for transfers to diskette.

The MCZ 1/35 bootstraps directly from DFS. The 4K monitor proms for this system contain the system resident portion of the DFS driver and the system bootstrap. ZDOS is supplied on the disk for use with optional floppy disk drives and is linked to run at the top of memory. There is a separate entry point for the FLOPPY driver at two greater than the entry point for ZDOS. When activating ZDOS, FLOPPY should be activated with an address in the same manner as was described above for DISK, i.e.

```
ACTIVATE $ZDOS;X * $FLOPPY E002
```

Command files for these systems are normally linked at 2A00.

7.2 DFS OPERATION

Except where noted otherwise, the information contained in Chapter 6 regarding ZDOS also applies to DFS.

The following are the major differences between the two file systems:

1) The DFS descriptor record differs from the ZDOS descriptor record:

Bytes 0-3	Unused
Bytes 4-7	File ID
Bytes 8-10	Pointer to directory sector holding entry for this file
Bytes 11-13	Pointer to first data record of file
Bytes 14-16	Pointer to last data record of file
Byte 17	File type and subtype
Bytes 18-19	Record count
Bytes 20-21	Record length
Bytes 22-23	Block length - currently unused, and set to be same as record length
Byte 24	File properties
Bytes 25-26	Starting execution address for procedure type files (entry point)
Bytes 27-28	Number of bytes in last record
Bytes 29-36	Date of creation
Bytes 37-44	Date of last modification
Bytes 45-132	Undefined by DFS - used by RIO for procedure files
Bytes 133-511	Available for programmer definition

2) While ZDOS accepts records of several lengths (multiples of 128 (80H) bytes), the record length on DFS must be equal to the physical sector length (512, or 200H bytes). A request to set it to any other record length results in an invalid attribute error (code D2H). However, a request for record length 0 is treated as a request for the default, or 200H.

3) Because some of the disks to be interfaced to this system have more than 65535 sectors, three bytes are used for all disk addresses. This affects

programs which access the directory, because there are now 3-byte pointers at the end of each entry instead of 2-byte pointers. To facilitate differentiation between the two formats, the DFS directory file is subtype 1.

4) There are three commands which apply exclusively to DFS disks. They are described in the following sections of this manual:

Command	Section
DISK.FORMAT	5.16
DISK.REPAIR	5.17
DISK.STATUS	5.18

5) For systems with floppy disks, software updates will be supplied on floppy diskettes as required. For systems without diskette drives, software updates are supplied on disk cartridges at an additional cost.

6) The GET/SAVE Debug commands have not yet been implemented for DFS hard disks.

7) The user and system operator should be familiar with the disk manufacturer's recommended operations procedures provided in separate manuals. These paragraphs are not intended to substitute for those documents. Nevertheless, we will briefly discuss disk start-up and shut-down procedures. These procedures apply to the Caelus Model 206R front-loading cartridge disks. Similar procedures would apply to other drives of the same class.

   Prior to bootstrapping the system, turn power on to the disk drive and disk controller. After a delay of perhaps 10-15 seconds, the "stop" light on the disk front panel will be illuminated. At this time open the door and slide in the cartridge to be used for the session. Close the door and turn the run/stop switch to run. After a delay of about a minute, the ready light will come on. At this point the bootstrap file from the disk is read into the controller memory. The disk is now ready for operation. Bootstrap the system.

To change disk cartridges during a session, turn the run/stop switch to stop, wait for the stop light, open the door, remove the cartridge and insert the new one, close the door and turn the switch to run, and wait for the ready light. The door is always locked except when the stop light is on. Also, a cartridge must be in place and the door closed to spin the disk up.

Any time there is a change in disks or a change in the ready status, give an Initialize (I) command. This is necessary to make the controller aware of the new status.

When shutting down for a period of time, as overnight, it is best to remove the cartridge and close the door, to prevent dust from gathering in either the cartridge or the drive.

7.3 SOFTWARE ORGANIZATION

The Disk File System driver is divided into two parts to correspond to the division of the hardware interface. One part is resident on the host system and contains the message interface link to communicate with the controller. It is used to pass I/O requests to the controller and to send and receive data.

The second, and larger part, is the software which resides in the memory of the intelligent controller. Most of this is loaded from the disk whenever the controller is reset. The software in ROM includes the initial start-up and serial message interface logic, the bare disk controller, which gives access to the disk by sector number for reads and writes, and the self-bootstrapping logic to read the rest of the file system from the disk. The disk file system, which is an adaptation of the ZDOS-II file system, and an interface for handling the request vector across the remote serial interface, are loaded from the disk.

There are two entry points to the resident software, each of which takes a standard I/O vector. One, DFS, causes the vector to be taken as a request to the Disk File System on the controller, and the other, DISK (located at address of DFS + 2), causes it to be taken as a request to the bare disk controller.

7.4 DFS ALLOCATION

7.4.1 SECTOR 0 FORMAT

As opposed to the floppy-disk system, which uses a RAM bit map, allocation is maintained as a list of free sectors. The first part of this list, plus the volume ID (Disk ID) and statistics on disk usage, are maintained on sector 0 of each unit. The format of that sector is as follows:

VOLUME ID -- bytes 1-100

MAXSIZ -- bytes 101-103 -- the total number of sectors on that unit, and thus 1 more than the highest accessible sector address.

MAXID -- bytes 104-107 -- the current highest file ID. This is incremented every time a new file is created.

ROOTPT -- bytes 108-110 -- the pointer to the root directory descriptor for the unit.

TFREE -- bytes 111-113 -- the total number of sectors remaining on the free list.

NERR -- bytes 114-116 -- the number of sectors unusable because of errors. This is generated during the read pass of the format routine.

NUSED -- bytes 117-119 -- total number of sectors on the unit which have been allocated.

NFREE -- bytes 120-121 -- an index into the free array to follow.

FREE -- bytes 122-421 -- an array of 100 3-byte pointers, each to an unallocated sector.

FMOD -- byte 422 -- internal use.

NSECS -- byte 423 -- number of sectors per track

SECMAP -- bytes 424-487 -- sector interlace map. Used by the disk controller to optimize track layout for access time.

ERRPTR -- bytes 488-490 -- pointer to a sector containing the list of sectors in error after the initial format.

BTPTR -- bytes 491-493 -- pointer to the descriptor record for the bootstrap file. Used by the bootstrap process to avoid a directory search.

The remainder of the sector is reserved for future use.

7.4.2 DFS ALLOCATION ALGORITHM

FREE[0] is a pointer to a sector whose contents are addresses of the next 100 elements of the free chain. To allocate a sector, decrement NFREE, and the new sector is FREE[NFREE]. If this element is zero, then there are no sectors left. If NFREE becomes 0, then read the contents of the designated sector into NFREE and FREE. To deallocate a sector, if NFREE is 100, copy NFREE and FREE to the deallocated sector and set NFREE to zero. Set FREE[NFREE] equal to the sector address and increment NFREE.

7.5 THE BARE DISK CONTROLLER

The bare disk controller is analogous to the subroutine FLOPPY in floppy-disk based systems. It accepts a standard RIO vector and reads or writes a sector of the disk. It accepts only a very limited set of requests, and requires a supplemental parameter vector specifying the disk address of the sector and the disk drive to be accessed. The drive is specified in the most significant three bits of the 3-byte disk address. The sectors are addressed by 3-byte integers. Sector 0 is cylinder 0, surface 0, sector 0. Since disks with various numbers of sectors per track, various numbers of surfaces, and various numbers of cylinders can all be used with the same interface, it is difficult to say on what cylinder, surface, and sector any given sector address would lie, or what the highest sector address is. However, with increasing sector address, the sector number varies most rapidly, then the surface, then the cylinder.

On drives which have one or more fixed platters and one removable platter, the removable platter is considered to be a different physical drive than, and thus to have a different sector address space from, the fixed platter or platters. If there are four drives on a controller, the fixed platters on them will be referenced as drives 0, 2, 4, and 6. It is not necessary that any drive have a removable platter, but if it does, the removable platter is referenced as drive n+1, where n is the drive by which the fixed platter is referenced. It is also not necessary that multiple drives be of the same configuration; i.e., the drives need not be in fixed-removable platter pairs.

The typical installation will have a single drive with one fixed platter, one removable platter, 12 sectors per track and 406 tracks, and will be connected as drive 0. This means that drive 0, sector 0 will address sector 0, surface 0, and cylinder 0 of the fixed platter. Sector 1 will access sector 1, surface 0, cylinder 0. Sector 12 will access sector 0, surface 1,

cylinder 0. Sector 23 will access sector 11, surface 1, cylinder 0. Sector 1000 will access sector 4, surface 1, and cylinder 41. The highest addressable sector will be 9743.

The same addresses on drive 1 would access sectors in the same relative position, but on the removable cartridge instead of the fixed platter.

The requests which the bare disk controller will handle are as follows:

INITIALIZE (00) -- initializes all controls and issues a recalibrate request to each disk. This causes each head to move to cylinder 0.

READ BINARY (0AH) -- transfers data from the disk starting at the requested sector until the byte count is satisfied, then continues to the end of the sector. The sector headers are not transferred.

WRITE BINARY (0EH) -- transfers data to the requested sector. The sector header of the sector preceding the selected one is read and checked for consistency before the sector is written. The first ten bytes of the data transfer area are the backward pointer, the forward pointer, and the file ID of the sector. They will be written in the sector header, not in the data area. Because of the necessity to provide a new set of pointers and file ID for each subsequent sector, only one sector can be written at a time.

READ HEADER (32H) -- The 24 bytes of the sector header of the requested sector are transferred back to the user. These are as follows:

byte 1 -- flag byte, always 80H
byte 2 -- cylinder address, low order
byte 3 -- cylinder address, high order
byte 4 -- surface
byte 5 -- sector
byte 6-8 -- backward pointer
byte 9-11 -- forward pointer
byte 12-13 -- data length, always 200H
byte 14-17 -- file ID
byte 18-19 -- header CRC
byte 20-24 -- zero

WRITE WITHOUT PRECHECK (34H) -- performs exactly as a write request, but skips the check of the header of the preceding sector. This permits initial formatting of the disk as well as repair of a damaged format.

7.6 CONTROLLER BOOTSTRAP OPERATION

The bootstrap of the controller occurs after the system is reset. At reset time a flag is set indicating that the controller is awaiting bootstrap; any request from the host system prior to controller bootstrap is answered by returning with a device not ready (Error C2) completion code.

All control variables and control ports are initialized. At this time, if drive 1 is ready, the reading of the bootstrap starts. If it is not ready, the STATUS PIO is set up to give an interrupt when drive 1 becomes ready, and transfer is made up to the wait-for-request loop. When the interrupt signals that drive 1 has entered the ready state, the reading of the bootstrap starts.

When reading the bootstrap, interrupts from the SIO are disabled to prevent any interference with the disk operation. Any request messages received during this time are ignored, causing them to be repeated until they are acknowledged.

The first step in reading the bootstrap is to read sector 0 of drive 1 (the removable cartridge) into a buffer. (If an error occurs at any point in reading the bootstrap from drive 1, the process is repeated on drive 0.) A variable in block 0, BTPTR, gives a pointer to the descriptor record of the bootstrap file. If this pointer is null, the bootstrap fails on drive 1, and bootstrap from drive 0 is initiated.

This descriptor record is read into the buffer. For each segment listed in the descriptor record, successive data records are read into a buffer and transferred to the correct address in the controller memory. As each data record is read in, its back pointer is checked against the address of the previous record. Any failure to compare causes failure of the bootstrap read on that drive.

When the last segment has been completely read, the flag is cleared indicating that the bootstrap has completed. The starting address from the descriptor is placed in a variable for indirect jumping on receipt of a DFS request.

Should the bootstrap read from drive 1 fail to complete for any reason (disk error or failure of a required condition), the entire process will be repeated on drive 0. Should it also fail on drive 0, it will again try drive 1, continuing to alternate between drives until the condition corrects itself or the operator turns off the disk.

7.7 SYSTEM BOOTSTRAPPING on the MCZ-1/35

Since the entire file system is available in ROM at bootstrap time, bootstrapping consists of issuing I/O requests to DFS, as follows:

1) Initialize
2) Open named file */OS for input, returning all attributes
3) For each segment, issuing read binary requests for the data address and data length indicated in the segment table.
4) Close file

and jumping to the entry point given by the attributes.

APPENDIX A

RIO/ZDOS/DFS ERROR CODES

RIO

Completion Code	Meaning
40	Invalid Drive Name
41	Invalid or Inactive Device
42	Invalid Unit
43	Memory Protect Violation
44	Missing or Invalid Operand(s)
45	System Error
46	Illegal File Name
47	Non-existent Command
48	Illegal File Type
49	Program Abort
4A	Insufficient Memory
4B	Missing or Invalid File Properties
4C	I/O Error (IY->Vector)

ZDOS/DFS

Completion Code	Meaning
80	Operation Complete
81	Directory Format Error
82	Scratch File Created
83	File Name Truncated
84	Attribute List Truncated
C1	Invalid Operation (Request)
C2	Device Is Not Ready
C3	Write Protection
C4	Sector Address Error
C5	Seek Error
C6	Data Transfer Error
C7	File Not Found
C9	End of File Error
CA	Pointer Check Error
CB	File Not Open
CCwww	Unit Already Active (Open)
CD	Assign Buffer Full
CE	Invalid Drive Specification
CF	Logical Unit Table Full (>16 Open)

D0	Duplicate File
D1	Diskette ID Error
D2	Invalid Attributes
D3	Disk Is Full
D4	File Not Found in Proper Directory Record
D5	Beginning of File Error
D6	File Already Open on Other Unit
D7	Invalid Rename to Scratch File
D8	File Locked (Attempt to Change Attributes)
D9	Invalid Open Request
DA	Insufficient Memory for Allocation Maps

APPENDIX B

RIO COMMAND SYNTAX SUMMARY

NAME	PARAMETERS	REFERENCE
ACTIVATE	device_name [address]	5.1
Allocate	low_boundary high_boundary block_size	5.2
Brief		5.3
CAT	(match_string | T=type | P=props | D=drive | F=format | L=listing_disposition | DATE rel date | CDATE rel date)*	5.4
Close	u|*	5.5
COMPARE	file_1 file_2	5.6
COPY	file_1 file_2 (A | U | O | RL=record length | T=type)*	5.7
COPY.DISK	[s_drive TO d_drive] [V]	5.8
COPYSD	file_name	5.9
DATE	[yymmdd]	5.10
DEACTIVATE	device_name	5.11
DEAllocate	block_address block_size	5.12
Debug		5.13
DEFINE	(unit file_name | unit device_name | unit * | *)+ [A | O | U | I | NF | NO]	5.14
DELETE	(match_string | T=type | P=props | D=drive | Q=query | DATE rel date | CDATE rel date)*	5.15
DISK.FORMAT	(S | D=drive | ID='disk_name' | Q=query)*	5.16

DISK.REPAIR	DFS_drive_number level_number	5.17
DISK.STATUS	[0|1 ... 6|7]	5.18
DISPLAY		5.19
DO	command_file [parameter list]	5.20
DUMP	file_name [m[n]]	5.21
ECHO	string	5.22
ERROR	[error_code|*]	5.23
ERRORS		5.24
EXTRACT	file_name	5.25
Force	command parameter_list	5.26
FORMAT	(S | D=drive | ID='disk_name' | Q=query)*	5.27
HELP	(key_word|*)*	5.28
IMAGE	filename (first_location last_location)+ [E=entry point] [RL=record length] [ST=stack size]	5.29
Initialize	[device_name] [parameter list]]	5.30
LADT		5.31
MASTER	[device_name]	5.32
MOVE	(match_string | T=type | P=props | F=format | D=destination_device | S=source_device | L=listing_disposition | Q=query | DATE rel date | CDATE rel date)*	5.33
PAUSE		5.34
Release		5.35

RENAME	(oldfile newfile | device:drive ID='new_disk_name')*	5.36
RESTORE_TABS	file_name	5.37
SAVE_TABS	file_name	5.38
SET	(CHRDEL=c | LINDEL=c | NULLCT=n | SPEED=nn | LFCNT=n | TABSIZE=n | ECHO ON | ECHO OFF | AUTOLF ON | AUTOLF OFF | PROPERTIES OF file_name TO plist | TYPE OF file_name TO type | SUBTYPE OF file_name TO subtype | ENTRY_POINT OF file_name TO nn | LOW_ADDRESS OF file_name TO | HIGH_ADDRESS OF file_name TO | STACK_SIZE OF file_name TO nn | BYTE_COUNT OF file_name TO nn)*	5.39
STATUS	[0|1 ... 6|7]	5.40
Verbose		5.41
Xeq	[* | nn [parameter_list]]	5.42
:	expression	5.43

APPENDIX C

RIO SYSTEM CONSTANTS

The following are the current values of RIO symbols. In some cases, however, address values vary from those listed below. To be certain you are using the values appropriate to your system, check the NOTE.TO.USER file on your system disk.

Symbol	MCZ Address	ZDS Address
BRKFLG	13CD	0FC4
BRKRTN	13CE	0FC5
CHRDEL	13CC	0FC3
CONIBF	1189	0D8A
CONIVC	1293	0ECE
CONOBF	1103	0D04
CONOVC	1288	0EC3
DATE	13AB	0FA2
DEBUG	0BFA	0BFA
DISK	0BFD	0BFD
ENTRY	17DE	13DE
ERCODE	13BD	0FB4
EXTRET	13BE	0FB5
FLOPPY	0BFD	0BFD
INITIALIZATION COMMAND AREA	18DF	14DF
INPTR	13C4	0FBB
INTERRUPT VECTOR	1300	0F00
LINDEL	13CB	0FC2
MEMMGR	1409	1009
NULLCT	13C8	0FBF
OUTPTR	13C6	0FBD
PROMPT	13CA	0FC1
PCON	0BE8	0BEE
SYSFLG	140E	100E
SYSTEM	1403	1003
SYSTEM REENTRY POINT	1400	1000

APPENDIX D

CONVERTING FILES TO RIO FORMAT

Files which have been created by MCZ or ZDS 2.1 System software (hereafter reference is made only to OS 2.1) are not compatible with RIO format. File conversion can be effected in two ways: a series of RIO commands can convert one or more files, or a Zilog utility program can be used which converts all files on a given diskette. In either case, the conversion is non-destructive, that is, the converted file always resides on a second disk, leaving the OS 2.1 type disk unaffected.

To manually convert one or more files to RIO format, enter the following command sequence:

```
%ACTIVATE $ZDOSI      ;Activates the OS 2.1 ZDOS emulator
%COPY,                ;Load, but do not execute, COPY
```

At this point, the diskette containing the file or files to be converted is inserted in one drive and the formatted RIO diskette on which the converted file is to reside is inserted in another. For this example, drive 2 has the RIO diskette, and drive 0 has the OS 2.1 type diskette.

```
%I;I $ZDOSI           ;Initialize the disk allocation maps
%XEQ * $ZDOSI:0/TEMP.S 2/TEMP.S
```

The above two commands may be repeated as necessary. The INITIALIZE command need be executed only when the diskettes are changed. The last command causes execution of the last loaded file, which in this case is COPY. (If a formatted RIO disk was in drive 0 with the COPY command on it, the user can alternatively enter 'COPY..!.) The first parameter is the source file name; note that it will always be qualified with '$ZDOSI' indicating that the OS 2.1 ZDOS emulator (ZDOSI) is the device on which the file exists. The second parameter is the destination file name, which does not have to be the same as the source file name; it may

or may not be qualified with '$ZDOS' as long as ZDOS is the master device.

After file conversion and verification of conversion is concluded, the following command will free the memory occupied by ZDOSI.

```
%DEACTIVATE $ZDOSI    ;Remove ZDOSI from Active Device Table
```

Alternatively, all files on a diskette may be converted using the utility program CONVERT. Execution of that command results in the prompt message,

```
INSERT DISKS (OLD DISK IN DRIVE 2, FORMATTED RIO DISK
IN DRIVE 0)
TYPE RETURN (OR 'Q' TO TERMINATE)
```

The OS 2.1 diskette is inserted into drive 2 and the RIO diskette is inserted into drive 0, after which carriage return (or 'Q' to abort) is entered. The names of the new RIO files will be the OS 2.1 type name concatenated with the file type as an extension. For example, the 'C' type file 'ABC' would be named 'ABC.C' on the RIO diskette. There is no provision to rename files as they are converted.

After all files on the OS 2.1 diskette are converted, the prompt message is repeated, at which time diskettes may be changed.

APPENDIX E

ALTERING DEFAULT RIO

The file OS contains the resident RIO programs plus the default system console driver. Altering the file consists of GETting OS from the 3K Monitor, making the desired modifications and SAVing it. The examples will use MCZ addresses; for ZDS addresses, refer to Appendix C.

Example 1: MODIFY SYSTEM FLAG TO INHIBIT EXTERNAL INITIALIZATION

```
%SET PROPERTIES OF 0/OS TO *  ;remove write protection
                               from file OS
%D                            ;enter 3K Monitor Debugger
>GET 0/OS                     ;load file OS from drive 0
>D 140E                       ;display and modify SYSFLG
140E 04 0 Q                   ;turn off EXTERNAL
                               INITIALIZATION FLAG
>SAV 0/OS 1400 2BFF E=17DE RL=400
                              ;save new OS
>OS                           ;bootstrap new OS
RIO
%SET PROPERTIES OF 0/OS to SW ;restore protection to file OS
```

Example 2: MODIFY SYSTEM EXTERNAL INITIALIZATION COMMAND AREA TO EXECUTE THE COMMAND 'BASIC' (ONLY) ON EXTERNAL INITIALIZATION

```
%SET PROPERTIES OF 0/OS TO *  ;remove write protection
%D                            ;enter 3K Monitor Debugger
>GET 0/OS                     ;load file OS from drive 0
>D 140E                       ;turn on external initialization
                               flag
140E 00 4 Q
>D 18DF                       ;display and open initialization
                               message area
18DF  44  42                  ;Change to 'B'
18E0  4F  41                  ;'A'
18E1  20  53                  ;'S'
18E2  30  49                  ;'I'
18E3  2F  43                  ;'C'
```

```
18E4  4F  0D Q                    ;carriage return and quit
>SAV 0/OS 1400 2BFF E=17DE RL=400
                                  ;save new OS
>OS                               ;bootstrapping at this point
                                   will result in execution
                                   of the file BASIC
```

APPENDIX F

I/O REQUEST VECTOR FORMAT

and

I/O REQUEST CODES

I/O Request Vector Format

Byte	Contents
0	Logical Unit Number
1	Request Code
2-3	Data Transfer Address
4-5	Data Length
6-7	Completion Address
8-9	Error Return Address
A	Completion Code
B-C	Supplemental Parameter Vector Address

ZDOS/DFS Supplemental Parameter Vector

0	Flag Byte (ASSIGN), Open Type (OPEN)
1	Drive Designation
2	File Name Length
3	File Name

I/O Request Codes

0	Initialize
2	Assign
4	Open
6	Close
8	Rewind
A	Read Binary

C	Read Line
E	Write Binary
10	Write Line
12	Write Current
14	Write Direct
16	Delete
18	Delete Remaining
1A	Erase File
1C	Read and Delete
1E	Read Current
20	Read Previous
22	Read Direct
24	Skip Forward
26	Skip Backward
28	Skip to End
2A	Rename
2C	Update
2E	Set Attributes
30	Query Attributes
40	Read Status
42	Write Status
44	Deactivate

APPENDIX G

PROGRAM EXAMPLES

Following are sample programs which the user is encouraged to edit, assemble, link, and execute. They illustrate some of the concepts introduced in previous sections, including console I/O, parameter string processing, and file I/O.

In each case the following commands may be used to edit, assemble, and link the example program:

```
%EDIT filename.S      ;invoke the editor
EDIT 1.6              ;the file does not already
NEW FILE              ;exist, so it is created
INPUT                 ;editor automatically enters
                      ;input mode



>QUIT
%ASM filename (X)     ;assemble (w/cross reference option)
ASM 5.7
PASS 1 COMPLETE
0 ASSEMBLY ERRORS
ASSEMBLY COMPLETE
%LINK $=4400 PRINT    ;and link
LINK 1.5
LINK COMPLETE
%
```

All system addresses are given for MCZ. Refer to Appendix C for ZDS equivalents.

```
                                EXAMPLE1.MCZ
 LOC   OBJ CODE M STMT SOURCE STATEMENT                          ASM 5.7

                      1  ;  EXAMPLE 1 -- MAKE A SYSTEM CALL TO PRINT A MESSAGE
                      2
                      3
                      4
0000   FD210800 R     5          LD      IY,AVEC          ; IY -> I/O VECTOR
0004   CD0314         6          CALL    SYSTEM
0007   C9             7          RET
                      8
                      9
                     10  AVEC:
0008   02            11  AVLUN:  DEFB    CONOUT           ; LOGICAL UNIT NUMBER
0009   10            12  AVREQ:  DEFB    WRTLIN           ; REQUEST CODE
000A   1300      R   13  AVDTA:  DEFW    MSG              ; DATA TRANSFER
                     14                                   ; ADDRESS
000C   2400          15  AVDL:   DEFW    LMSG             ; BYTE COUNT
000E   0000          16  AVCRA:  DEFW    0                ; COMPLETION RETURN
                     17                                   ; ADDRESS
0010   0000          18  AVERA:  DEFW    0                ; ERROR RETURN ADDRESS
0012   00            19  AVCC:   DEFB    0                ; COMPLETION CODE
                     20
                     21  ; EQUATES AND CONSTANTS
                     22
                     23  SYSTEM: EQU     1403H            ; SYSTEM ENTRY POINT
                     24  CONOUT: EQU     2                ; CONSOLE OUTPUT UNIT
                     25  WRTLIN: EQU     10H              ; WRITE LINE REQUEST
                     26                                   ; CODE
                     27
0013   454E4F52      28  MSG:    DEFM    'ENORMOUS CHANGES AT THE LAST MINUTE'
0036   0D            29          DEFB    0DH
                     30  LMSG:   EQU     $-MSG
                     31
                     32          END


CROSS REFERENCE                 EXAMPLE1.MCZ
SYMBOL  VAL M DEFN REFS

AVCC    0012 R   19
AVCRA   000E R   16
AVDL    000C R   15
AVDTA   000A R   13
AVEC    0008 R   10    5
AVERA   0010 R   18
```

```
SYMBOL   VAL M DEFN REFS

AVLUN   0008 R   11
AVREQ   0009 R   12
CONOUT  0002     24   11
LMSG    0024     30   15
MSG     0013 R   28   13   30
SYSTEM  1403     23    6
WRTLIN  0010     25   12
```

```
                                EXAMPLE2.MCZ
LOC   OBJ CODE      M STMT SOURCE STATEMENT                          ASM 5.7

                       1 ;EXAMPLE 2 --- MAKE A SYSTEM CALL TO COPY THE PARAMETER
                       2 ;               STRING TO CONOUT
                       3 ;
                       4
0000  2AC413           5          LD      HL,(INPTR)       ;ADDRESS OF THE
                       6                                   ;PARAMETER LIST
0003  113700      R    7          LD      DE,BUFFER        ;MOVE PARAMETER LIST
                       8                                   ;INTO BUFFER
0006  010000           9          LD      BC,0             ;KEEP A CHARACTER COUNT
0009  7E              10 SCAN:    LD      A,(HL)           ;NEXT CHARACTER IN
                      11                                   ;PARAMETER STRING
000A  FE3B            12          CP      ';'              ;TEST FOR END
000C  2808            13          JR      Z,ADDCHR
000E  FE0D            14          CP      0DH
0010  2804            15          JR      Z,ADDCHR
0012  EDA0            16          LDI                      ;MOVE CHARACTER AND INC
                      17                                   ;POINTERS
0014  18F3            18          JR      SCAN
                      19
0016  EB              20 ADDCHR:  EX      DE,HL
0017  360D            21          LD      (HL),0DH         ;COULD HAVE BEEN A ';'
0019  0B              22          DEC     BC
                      23
001A  79              24          LD      A,C              ;GET STRING LENGTH
001B  2F              25          CPL
001C  4F              26          LD      C,A
001D  2F              27          CPL
001E  47              28          LD      B,A
001F  03              29          INC     BC
                      30
0020  ED433000    R   31          LD      (AVDL),BC        ;DATA LENGTH
                      32
0024  FD212C00    R   33          LD      IY,AVEC
0028  CD0314          34          CALL    SYSTEM           ;MAKE THE SYSTEM CALL
                      35                                   ;TO WRITE IT
002B  C9              36          RET
                      37
                      38
                      39 AVEC:
002C  02              40 AVLUN:   DEFB    CONOUT           ;CONSOLE OUTPUT
002D  10              41 AVREQ:   DEFB    WRTLIN           ;WRITE LINE
002E  3700        R   42 AVDTA:   DEFW    BUFFER           ;DATA TRANSFER ADDRESS
0030  0000            43 AVDL:    DEFW    0                ;DATA LENGTH
0032  0000            44 AVCRA:   DEFW    0                ;COMPLETION RETURN
```

```
 LOC   OBJ CODE M STMT SOURCE STATEMENT                          ASM 5.7

                    45                                          ;ADDRESS
0034   0000         46  AVERA:  DEFW    0                      ;ERROR RETURN ADDRESS
0036   00           47  AVCC:   DEFB    0                      ;COMPLETION CODE
                    48
                    49
                    50  SYSTEM: EQU     1403H                  ;SYSTEM ENTRY POINT
                    51  CONOUT: EQU     2                      ;CONSOLE OUTPUT UNIT
                    52  WRTLIN: EQU     10H                    ;WRITE LINE REQUEST
                    53                                         ;CODE
                    54  INPTR:  EQU     13C4H                  ;PARAMETER LIST POINTER
                    55
0037                56  BUFFER: DEFS    80
                    57          END


CROSS REFERENCE                 EXAMPLE2.MCZ
SYMBOL  VAL M DEFN REFS

ADDCHR 0016 R   20   13   15
AVCC   0036 R   47
AVCRA  0032 R   44
AVDL   0030 R   43   31
AVDTA  002E R   42
AVEC   002C R   39   33
AVERA  0034 R   46
AVLUN  002C R   40
AVREQ  002D R   41
BUFFER 0037 R   56    7   42
CONOUT 0002     51   40
INPTR  13C4     54    5
SCAN   0009 R   10   18
SYSTEM 1403     50   34
WRTLIN 0010     52   41
```

```
                                EXAMPLE3.MCZ
LOC   OBJ CODE  M STMT SOURCE STATEMENT                              ASM 5.7

                    1 ;PROGRAM 3 -- PRINT -- COPY AN ASCII FILE TO SYSLST
                    2 ;
                    3 ;
                    4 ;
                    5
                    6
                    7 ;MAKE A SYSTEM CALL TO FORMAT SUPPLEMENTAL PARAMETEF
                    8 ;VECTOR
                    9
0000  3E02          10          LD      A,ASSIGN        ;ASSIGN REQUEST
0002  325001    R   11          LD      (AVREQ),A
0005  3E04          12          LD      A,4             ;ON UNIT 4
0007  324F01    R   13          LD      (AVLUN),A
000A  AF            14          XOR     A
000B  325C01    R   15          LD      (SPVFB),A
000E  2AC413        16          LD      HL,(INPTR)      ;PARAMETER STRING
                    17                                  ;ADDRESS
0011  225101    R   18          LD      (AVDTA),HL
0014  215C01    R   19          LD      HL,SPV          ;-> SUPPLEMENTAL
                    20                                  ;PARAMETER VECTOR
0017  225A01    R   21          LD      (AVSVP),HL
001A  210000        22          LD      HL,0            ;CLEAR ERROR RETURN
                    23                                  ;ADDRESS
001D  225701    R   24          LD      (AVERA),HL      ;FOR NORMAL ERROR
                    25                                  ;RETURN
0020  FD214F01  R   26          LD      IY,AVEC
0024  CD0314        27          CALL    SYSTEM
0027  3A5901    R   28          LD      A,(AVCC)        ;COMPLETION CODE
002A  CB77          29          BIT     6,A             ;ERROR
002C  C2D700    R   30          JP      NZ,ERROR
                    31
                    32
                    33 ;OPEN THE FILE AND TEST FILE TYPE
                    34
                    35
002F  3E04          36          LD      A,OPEN          ;OPEN REQUEST
0031  325001    R   37          LD      (AVREQ),A
0034  3E00          38          LD      A,OPNINP        ;FOR INPUT
0036  325C01    R   39          LD      (SPVOR),A
0039  3EFF          40          LD      A,-1            ;UNIT PREVIOUSLY
                    41                                  ;ASSIGNED
003B  325E01    R   42          LD      (SPVFNL),A
003E  217F01    R   43          LD      HL,ASVFT        ;REQUEST SOME OF THE
                    44                                  ;DESCRIPTOR RECORD
```

```
0041  225101    R   45          LD      (AVDTA),HL
0044  210100        46          LD      HL,LASV
0047  225301    R   47          LD      (AVDL),HL
004A  CD0314        48          CALL    SYSTEM
004D  3A5901    R   49          LD      A,(AVCC)        ;COMPLETION CODE
0050  CB77          50          BIT     6,A             ;ERRORS?
0052  C2D700    R   51          JP      NZ,ERROR
                    52
                    53
                    54  ;FILE IS OPEN TEST FILE TYPE, COPY TO SYSLST
                    55
                    56
0055  3A7F01    R   57          LD      A,(ASVFT)       ;FILE TYPE
0058  E6F0          58          AND     0F0H            ;STRIP SUBTYPE
005A  FE20          59          CP      ASCII           ;BETTER BE ASCII
005C  2808          60          JR      Z,PRT100
005E  3E48          61          LD      A,ILLFT         ;OR ILLEGAL FILE TYPE
0060  32BD13        62          LD      (ERCODE),A
0063  C3F900    R   63          JP      CLOSEF
                    64
                    65
                    66  ;FILE IS OF TYPE ASCII
                    67
                    68
                    69  PRT100:
0066  3E00          70          LD      A,0             ;ALLOCATE
0068  210000        71          LD      HL,0
006B  11FFFF        72          LD      DE,-1           ;LOCATE LONGEST BUFFER
006E  01FFFF        73          LD      BC,-1
0071  CD0914        74          CALL    MEMMGR
0074  0E00          75          LD      C,0
0076  78            76          LD      A,B
0077  E6F0          77          AND     0F0H            ;SEE IF AVAILABLE
                    78                                  ;SPACE >= 1000H
0079  2009          79          JR      NZ,GETIT        ;IF SO, ALLOCATE
                    80                                  ;IT
007B  CDF900    R   81          CALL    CLOSEF
007E  3E4A          82          LD      A,INSMEM        ;OTHERWISE, OUTPUT
                    83                                  ;INSUFF. MEM. ERROR,
                    84                                  ;CLOSE FILE AND
                    85                                  ;RETURN
0080  32BD13        86          LD      (ERCODE),A
0083  C9            87          RET
0084  47            88  GETIT:  LD      B,A
0085  AF            89          XOR     A
0086  CD0914        90          CALL    MEMMGR
0089  224B01    R   91          LD      (BUFFER),HL     ;SAVE BEGINNING
```

```
LOC   OBJ CODE M STMT SOURCE STATEMENT                          ASM 5.7

                    92                                          ;ADDRESS
008C  ED434D01  R   93          LD      (BUFSIZ),BC             ;SAVE SIZE OF BUFFER
                    94
0090  225101    R   95          LD      (AVDTA),HL              ;INITIALIZE DATA
                    96                                          ;TRANSFER ADDRESS
0093  ED435301  R   97          LD      (AVDL),BC               ;LOAD BUFFER SIZE
                    98
                    99
                   100  ;READ NEXT BUFFER LOAD
                   101
                   102
0097  3E0A          103  READ:  LD      A,RDBIN                 ;READ BINARY
0099  325001    R   104          LD      (AVREQ),A
009C  3E04          105          LD      A,4                     ;FROM UNIT 4
009E  324F01    R   106          LD      (AVLUN),A
00A1  FD214F01  R   107          LD      IY,AVEC
00A5  CD0314        108          CALL    SYSTEM
00A8  3A5901    R   109          LD      A,(AVCC)                ;READ ERROR?
00AB  CB77          110          BIT     6,A
00AD  280B          111          JR      Z,WRITE
00AF  FEC9          112          CP      EOF                     ;YES, HAS IT AN END
                   113                                          ;OF FILE?
00B1  2024          114          JR      NZ,ERROR
00B3  2A5301    R   115          LD      HL,(AVDL)
00B6  7C            116          LD      A,H                     ;YES, ANY DATA
                   117                                          ;TRANSFERRED?
00B7  B5            118          OR      L
00B8  283F          119          JR      Z,CLOSEF                ;IF NOT, CLOSE FILE
                   120
                   121
                   122  ;WRITE A BUFFER LOAD TO SYSLST
                   123
                   124
00BA  3E0E          125  WRITE: LD      A,WRTBIN                ;WRITE BINARY
00BC  325001    R   126          LD      (AVREQ),A
00BF  3E03          127          LD      A,SYSLST                ;ON SYSLST
00C1  324F01    R   128          LD      (AVLUN),A
00C4  3A5901    R   129          LD      A,(AVCC)                ;SAVE COMPLETION CODE
00C7  F5            130          PUSH    AF
                   131
                   132                                          ;DATA TRANSFER ADDRESS
                   133                                          ;AND DATA LENGTH WERE
                   134                                          ;SET BY THE READ
                   135                                          ;OPERATION
                   136
00C8  CD0314        137          CALL    SYSTEM
                   138
```

```
00CB   F1          139          POP     AF              ;RESTORE READ
                   140                                  ;COMPLETION CODE
00CC   FEC9        141          CP      EOF             ;DID LAST READ REQUEST
                   142                                  ;REACH END OF FILE?
00CE   2829        143          JR      Z,CLOSEF
                   144
                   145
00D0   3A5901   R  146          LD      A,(AVCC)        ;HAS WRITTEN
                   147                                  ;SUCCESSFULLY?
00D3   CB77        148          BIT     6,A
00D5   28C0        149          JR      Z,READ
                   150
                   151
                   152  ;AN ERROR HAS OCCURRED, PRINT
                   153  ;MESSAGE, CLOSE FILE, AND RETURN
                   154
                   155
00D7   3E0E        156  ERROR:  LD      A,WRTBIN        ;WRITE BINARY
00D9   325001   R  157          LD      (AVREQ),A
00DC   3E02        158          LD      A,CONOUT        ;TO CONSOLE OUTPUT UNIT
00DE   324F01   R  159          LD      (AVLUN),A
00E1   214801   R  160          LD      HL,EMSGN        ;CONVERT ERROR CODE TO
                   161                                  ;ASCII
00E4   3A5901   R  162          LD      A,(AVCC)
00E7   CD2401   R  163          CALL    BTOHEX
                   164
00EA   213E01   R  165          LD      HL,EMSG         ;PRINT MESSAGE
00ED   225101   R  166          LD      (AVDTA),HL
00F0   210D00      167          LD      HL,LEMSG
00F3   225301   R  168          LD      (AVDL),HL
00F6   CD0314      169          CALL    SYSTEM
                   170
                   171
00F9   3E06        172  CLOSEF: LD      A,CLOSE         ;CLOSE FILE
00FB   325001   R  173          LD      (AVREQ),A
00FE   3E04        174          LD      A,4             ;ON UNIT FOUR
0100   324F01   R  175          LD      (AVLUN),A
0103   210000      176          LD      HL,0
0106   225101   R  177          LD      (AVDTA),HL      ;DON'T UPDATE
                   178                                  ;DESCRIPTOR RECORD
0109   225301   R  179          LD      (AVDL),HL
010C   225A01   R  180          LD      (AVSVP),HL
010F   CD0314      181          CALL    SYSTEM
                   182
                   183                                  ;DEALLOCATE THE
                   184                                  ;ALLOCATED SPACE
                   185
```

```
 LOC   OBJ CODE M STMT SOURCE STATEMENT                              ASM 5.7


0112   2A4B01   R  186          LD      HL,(BUFFER)
0115   7C          187          LD      A,H             ;WAS BUFFER ALLOCATED?
0116   B4          188          OR      H
0117   3E01        189          LD      A,1             ;DEALLOCATE
0119   ED4B4D01 R  190          LD      BC,(BUFSIZ)
011D   C40914      191          CALL    NZ,MEMMGR
0120   32BD13      192          LD      (ERCODE),A
                   193
0123   C9          194          RET
                   195
                   196
                   197  ;B T O H E X --- CONVERT 8 BITS OF REG A TO HEX ASCII,
                   198  ;               STORE AT (HL)
                   199
0124   F5          200  BTOHEX: PUSH    AF              ;SAVE A
0125   1F          201          RRA
0126   1F          202          RRA
0127   1F          203          RRA
0128   1F          204          RRA
0129   CD3201   R  205          CALL    HBTHEX          ;CONVERT HIGH ORDER
                   206                                  ;4 BITS
012C   F1          207          POP     AF              ;RESTORE A
012D   23          208          INC     HL
012E   CD3201   R  209          CALL    HBTHEX          ;CONVERT LOW ORDER 4
                   210                                  ;BITS
0131   C9          211          RET
                   212
                   213
                   214  ;H B T H E X --- CONVERT 4-BIT BINARY LOW ORDER 4 BITS
                   215  ;               OF REG A TO HEX ASCII CHARACTER AT (HL)
                   216
0132   E60F        217  HBTHEX: AND     0FH             ;MASK OFF LOW ORDER 4
                   218                                  ;BITS
0134   FE0A        219          CP      10              ;DECIMAL CHARACTER?
0136   3802        220          JR      C,HB10
0138   C607        221          ADD     A,7             ;NO
013A   C630        222  HB10:   ADD     A,30H
013C   77          223          LD      (HL),A
013D   C9          224          RET
                   225
                   226
                   227  CONOUT: EQU     2               ;CONSOLE OUTPUT UNIT
                   228  SYSLST: EQU     3               ;SYSTEM VOLUME OUTPUT
                   229                                  ;UNIT
                   230
                   231
                   232  ;I/O REQUEST CODES
```

```
                    233
                    234  RDBIN:   EQU     0AH             ;READ BINARY
                    235  WRTBIN:  EQU     0EH             ;WRITE BINARY
                    236  WRTLIN:  EQU     10H             ;WRITE LINE
                    237  ASSIGN:  EQU     2               ;ASSIGN
                    238  OPEN:    EQU     4               ;OPEN
                    239  CLOSE:   EQU     6               ;CLOSE
                    240
                    241  OPNINP:  EQU     0               ;OPEN TYPE: OPEN FOR
                    242                                   ;INPUT
                    243  EOF:     EQU     0C9H            ;EOF ERROR CODE
                    244  ILLFT:   EQU     48H             ;RIO ERROR CODE -
                    245                                   ;ILLEGAL FILE TYPE
                    246  INSMEM:  EQU     4AH             ;RIO ERROR CODE -
                    247                                   ;INSUFFICIENT MEMORY
                    248  ASCII:   EQU     20H             ;ASCII FILE TYPE
                    249
                    250
                    251  ;RIO ADDRESSES
                    252
                    253  INPTR:   EQU     13C4H           ;PARAMETER STRING
                    254                                   ;POINTER
                    255  ERCODE:  EQU     13BDH           ;ERROR CODE LOCATION
                    256  SYSTEM:  EQU     1403H           ;SYSTEM CALL ADDRESS
                    257  MEMMGR:  EQU     1409H           ;MEMORY MANAGER ADDRESS
                    258
013E  492F4F20      259  EMSG:    DEFM    'I/O ERROR '
0148                260  EMSGN:   DEFS    2
014A  0D            261           DEFB    0DH             ;ADD A CARRIAGE RETURN
                    262  LEMSG:   EQU     $-EMSG
                    263
014B  0000          264  BUFFER:  DEFW    0               ;READ/WRITE BUFFER
014D                265  BUFSIZ:  DEFS    2
                    266
                    267  AVEC:
014F                268  AVLUN:   DEFS    1               ;LOGICAL UNIT NUMBER
0150                269  AVREQ:   DEFS    1               ;REQUEST CODE
0151                270  AVDTA:   DEFS    2               ;DATA TRANSFER ADDRESS
0153                271  AVDL:    DEFS    2               ;DATA LENGTH
0155                272  AVCRA:   DEFS    2               ;COMPLETION RETURN
                    273                                   ;ADDRESS
0157                274  AVERA:   DEFS    2               ;ERROR RETURN ADDRESS
0159                275  AVCC:    DEFS    1               ;COMPLETION CODE
015A                276  AVSVP:   DEFS    2               ;SUPPLEMENTAL PARAMETER
                    277                                   ;VECTOR POINTER
                    278
                    279  SPV:                             ;THE SUPPLEMENTAL
```

```
 LOC   OBJ CODE M STMT SOURCE STATEMENT                              ASM 5.7

                   280                                          ;PARAMETER VECTOR
                   281  SPVOR:                                   ;OPEN REQUEST TYPE
015C               282  SPVFB:  DEFS    1                       ;ASSIGN REQUEST FLAG
                   283                                          ;BYTE
015D               284  SPVDRV: DEFS    1                       ;DRIVE DESIGNATION
015E               285  SPVFNL: DEFS    1                       ;FILE NAME LENGTH
015F               286  SPVFN:  DEFS    32                      ;FILE NAME
                   287
                   288
                   289  ;AREA IN WHICH TO MOVE THE DESCRIPTOR RECORD ON THE
                   290  ;FILE OPEN
                   291
017F               292  ASVFT:  DEFS    1
                   293  LASV:   EQU     $-ASVFT
                   294
                   295          END


CROSS REFERENCE                    EXAMPLE3.MCZ
SYMBOL   VAL M DEFN REFS

ASCII   0020     248   59
ASSIGN  0002     237   10
ASVFT   017F R   292   43   57  293
AVCC    0159 R   275   28   49  109  129  146  162
AVCRA   0155 R   272
AVDL    0153 R   271   47   97  115  168  179
AVDTA   0151 R   270   18   45   95  166  177
AVEC    014F R   267   26  107
AVERA   0157 R   274   24
AVLUN   014F R   268   13  106  128  159  175
AVREQ   0150 R   269   11   37  104  126  157  173
AVSVP   015A R   276   21  180
BTOHEX  0124 R   200  163
BUFFER  014B R   264   91  186
BUFSIZ  014D R   265   93  190
CLOSE   0006     239  172
CLOSEF  00F9 R   172   63   81  119  143
CONOUT  0002     227  158
EMSG    013E R   259  165  262
EMSGN   0148 R   260  160
EOF     00C9     243  112  141
ERCODE  13BD     255   62   86  192
ERROR   00D7 R   156   30   51  114
GETIT   0084 R    88   79
HB10    013A R   222  220
```

```
SYMBOL  VAL M DEFN REFS


HBTHEX 0132 R  217  205  209
ILLFT  0048    244   61
INPTR  13C4    253   16
INSMEM 004A    246   82
LASV   0001    293   46
LEMSG  000D    262  167
MEMMGR 1409    257   74   90  191
OPEN   0004    238   36
OPNINP 0000    241   38
PRT100 0066 R   69   60
RDBIN  000A    234  103
READ   0097 R  103  149
SPV    015C R  279   19
SPVDRV 015D R  284
SPVFB  015C R  282   15
SPVFN  015F R  286
SPVFNL 015E R  285   42
SPVOR  015C R  281   39
SYSLST 0003    228  127
SYSTEM 1403    256   27   48  108  137  169  181
WRITE  00BA R  125  111
WRTBIN 000E    235  125  156
WRTLIN 0010    236
```

APPENDIX H

INTERNAL COMMAND TABLE CONTENTS

Debug
Initialize
Brief
Verbose
Xeq
Allocate
DEAllocate
Release
Force
Close
: (expression evaluation)

APPENDIX I

RIO MEMORY MANAGER

This appendix describes register contents before and after a MEMMGR call. Appendix C has the MCZ and ZDS addresses of MEMMGR. Example 3 of Appendix G includes calls to MEMMGR to allocate and deallocate buffer space.

ALLOCATE

Before MEMMGR CALL:
- A=0 (allocate)
  - HL = lower address boundary
  - DE = upper address boundary
  - BC = required size (bytes)

After MEMMGR CALL:
- A=80 (operation complete)
  - HL = beginning address of hole
  - DE = ending address of hole
  - BC = size of hole (bytes)
- A=4A (insufficient memory)
  - HL = beginning of largest hole within boundaries
  - BC = size of largest hole within boundaries (bytes)
    (if BC=0, then HL=undefined)

DEALLOCATE

Before MEMMGR CALL:
- A=1 (deallocate)
  - HL = beginning address of hole
  - BC = hole size (bytes)

After MEMMGR CALL:
- A=80 (operation complete)
- A=43 (not all blocks in area were allocated)

APPENDIX J

DESCRIPTOR RECORD OF PROCEDURE TYPE FILE

Byte #	
0..3	: Reserved for future expansion
4..5	: File ID - currently unused
6..7	: Pointer to directory sector holding entry for this file
8..9	: Pointer to first data record of file
10..11	: Pointer to last data record of file
12	: Type of file - see description under the OPEN request
13..14	: Record count
15..16	: Record length
17..18	: Block length - currently unused, and set to be same as record length
19	: Protection properties - see description in OPEN request
20..21	: Starting execution address for procedure files
22..23	: Number of bytes in the last record of the file
24..31	: Date of creation
32..39	: Date last written

40..40+4*n	: Where n is the number of segment descriptors (0 <= n <= 16) Each segment descriptor is 4 bytes-- the first 2 bytes are the starting address of the segment and the second 2 bytes are the length of the segment in bytes. After the the last segment descriptor is a null descriptor where each of the 4 bytes are zero.
122..123	: Lowest segment starting address (LOW_ADDRESS)
124..125	: Highest segment ending address rounded up to the end of the record (HIGH_ADDRESS)
126..127	: Stack size

APPENDIX K

ZDOS/DFS COMMAND SUMMARY

Name	Request Code	Ref
ASSIGN	02,03	6.2
CLOSE	06,07w	6.4
DELETE	16,17	6.9
DELETE REMAINING RECORDS	18,19	6.10
ERASE	1A,1B	6.11
INITIALIZE	00,01	6.1
OPEN	04,05	6.3
QUERY ATTRIBUTES	30,31	6.22
READ AND DELETE	1C,1D	6.12
READ BINARY	0A,0B	6.6
READ CURRENT	1E,1F	6.13
READ DIRECT	22,23	6.15
READ PREVIOUS	20,21	6.14
RENAME	2A,2B	6.19
REWIND	08,09	6.5
SET ATTRIBUTES	2E,2F	6.21
SKIP BACKWARD	26,27	6.17
SKIP FORWARD	24,25	6.16
SKIP TO END	28,29	6.18
UPDATE	2C,2D	6.20
WRITE BINARY	0E,0F	6.7
WRITE CURRENT	12,13	6.8

APPENDIX L

RELINKING RIO COMMANDS

Most RIO commands are linked to execute at the lowest available memory. Those which may be expected to be loaded concurrently under some circumstances (i.e., when executed from DO) are linked elsewhere. The object files for these commands (listed below) are provided on the system disk in order that they may be relinked to execute elsewhere to fit the user's needs.

DO
ECHO
PAUSE
IMAGE

The command files RELINK.MCZ.COMMAND and RELINK.ZDS.COMMAND are on the MCZ and ZDS system disks, respectively. These command files use the specified object file and link the command at the specified address using system object files on the system disk.

A command may be linked as follows:

```
DO RELINK.MCZ.COMMAND #1 #2 #3 #4 #5 #6 #7
```

or

```
DO RELINK.ZDS.COMMAND #1 #2 #3 #4 #5 #6 #7
```

where

#1 is the name of the command to be linked

#2 is the address the command is to be linked at

#3,#4,#5,#6,#7 are the optional link parameters (see RIO Relocating Assembler and Linker User's Manual for more details).

EXAMPLES

```
DO RELINK.MCZ.COMMAND DO 0C000

DO RELINK.ZDS.COMMAND IMAGE 0F000 RL=400
```

READER COMMENTS

Zilog

Your comments concerning this publication are important to us. Please take the time to complete this questionnaire and return it to Zilog.

Title of Publication: _____

Document Number: _____

Your Hardware Model and Memory Size: _____

Describe your likes/dislikes concerning this document:

Technical Information: _____

Supporting Diagrams: _____

Ease of Use: _____

Your Name: _____

Company and Address: _____

Your Position/Department: _____

CF-1031-02

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 475, CUPERTINO, CA.

POSTAGE WILL BE PAID BY ADDRESSEE

Zilog

Manager, Systems Publications
10460 Bubb Road
Cupertino, California 95014

DOCUMENT CHANGE NOTICE

Zilog

DATE: 10-31-79

DCN NUMBER: E3-0072-01, Rev. B

PUBLICATION NUMBER: 03-0072-01

TITLE: Z80 RIO Operating System User Manual

PREVIOUS DCNs BY NUMBER: E3-0072-01, Rev. A

EFFECTIVE DATE: 10-31-79

This document change notice provides changed pages for the publication specified above. These changed pages will remain in effect for subsequent releases unless specifically amended by another DCN or superseded by a publication revision. Replace the following pages in your manual with the enclosed pages:

31
47
85
99
105
107
A-1
B-1

Changes are indicated by a vertical line in the right-hand margin.

NOTE: Please file this DCN at the back of the manual to provide a record of changes.

DOCUMENT CHANGE NOTICE

Zilog

DATE: 02-13-79

DCN NUMBER:E3-0072-01, Rev. A

PUBLICATION NUMBER: 03-0072-01, Rev. A

TITLE: Z80-RIO Operating System Users Manual

PREVIOUS DCN's, BY NUMBER: None

EFFECTIVE DATE: 02-13-79

This Document Change Notice provides change pages for the publication specified above. These change pages will remain in effect for subsequent releases unless specifically amended by another DCN or superseded by a publication revision. The following pages are to be treated as described:

Replace Pages 55-60
Replace Pages 67-74
Insert Pages 98A and 98B after Page 98

Changes to text are indicated by a vertical line in the right margin, opposite the changed text portion.

NOTE: Please file this DCN at the back of the manual to provide a record of changes.

Zilog, Inc. 10460 Bubb Road, Cupertino, California 95014 Telephone (408)446-4666 TWX 910-338-7621

Printed in USA