intel®

INTELLEC® SERIES IV ISIS-IV USER'S GUIDE

Order Number: 121880-001

intel®

INTELLEC® SERIES IV ISIS-IV USER'S GUIDE

Order Number: 121880-001

A833/1182/1K Jay

REV.	REVISION HISTORY	DATE
-001	Original issue.	12/82

PREFACE

The *Intellec Series IV ISIS-IV User's Guide* provides an overview of the iNDX operating system, and specific operating instructions for the ISIS-IV user of the Series IV Development System.

This manual assumes you have read the *Intellec Series IV Microcomputer Development System Overview*, 121752, and the *Intellec Series IV Operating and Programming Guide*, 121753.

This manual contains six chapters and five appendixes:

- Chapter 1, "iNDX Overview," briefly describes the iNDX operating system of the Series IV, and introduces you to ISIS-IV, a subsystem of iNDX.
- Chapter 2, "File Creation and Management," describes iNDX files, some console user aids, and the ISIS-IV commands necessary to create, revise, and manage files.
- Chapter 3, "Use of ISIS-IV by Other Programs," describes the ISIS-IV system calls available for writing programs. The memory organization and allocation of Intellec, and parameters to system service routines are discussed.
- Chapter 4, "MON 85," describes the function and commands of the ISIS Executive Unit Monitor (MON 85).
- Chapter 5, "Interrupt Processing," describes the levels, the acceptance, and the removal of interrupts on the Series IV.
- Chapter 6, "ISIS-IV Error Codes and Messages," describes the processing and debugging of ISIS-IV errors. It also lists and explains all ISIS-IV error messages and indicates appropriate corrective action.
- Appendix A, "Summary of Error Messages," lists the error codes and messages issued by ISIS-IV and some non-resident system routines.
- Appendix B, "Summary of ISIS-IV Command Syntax," lists the ISIS-IV commands and syntax.
- Appendix C, "Monitor Command Summary," lists the commands and syntax for the ISIS Execution Unit Monitor (MON 85).
- Appendix D, "ASCII Codes," provides an ASCII Code list.
- Appendix E, "Hexadecimal-Decimal Conversion," provides a table for hexadecimal to decimal and decimal to hexadecimal conversion.

Related Publications

For further information on the Series IV Development System, refer to the following publications:

- *Intellec Series IV Microcomputer Development System Overview*, order number 121752
- *Intellec Series IV Operating and Programming Guide*, order number 121753
- *Intellec Series IV Pocket Reference*, order number 121760
- *Intellec Series IV ISIS-IV Pocket Reference*, order number 121890
- *DEBUG-88 User's Guide*, order number 121758

- *iAPX 88 Book*, order number 210200
- *iAPX 86,88 User's Manual*, order number 210201
- *iAPX 86,88 Family Utilities User's Guide*, order number 121616
- *MCS-80/85 Family User's Manual*, order number 121506
- *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems*, order number 121617
- *8080/8085 Floating-Point Arithmetic Library User's Manual*, order number 9800452
- *An Introduction to ASM86*, order number 121689
- *ASM86 Macro Assembler Operating Instructions*, order number 121628
- *ASM86 Language Reference Manual*, order number 121703

For information on the text editor that runs on ISIS, refer to the *ISIS-II CREDIT (CRT-Based Text Editor) User's Guide*, order number 9800902.

For the most complete and up-to-date list of software and hardware documentation, refer to the current *Literature Guide*, order number 802800. This publication provides references to such publications as PL/M-86 programming manuals, ASM80 and ASM86 manuals, Pascal-80 and Pascal-86 manuals, and FORTRAN-80 manuals. References to these manuals are also provided in *Intellec Series IV Microcomputer Development System Overview*, order number 121752.

Notational Conventions

UPPERCASE	Characters shown in uppercase must be entered in the order shown. You may enter the characters in uppercase or lowercase.
italic	Italic indicates a meta symbol that may be replaced with an item that fulfills the rules for that symbol. The actual symbol may be any of the following:
directory-name	Is that portion of a *pathname* that acts as a file locator by identifying the device and/or directory containing the *filename*.
filename	Is a valid name for the part of a *pathname* that names a file.
pathname	Is a valid designation for a file; in its entirety, it consists of a *directory* and a *filename*.
pathname1, pathname2, ...	Are generic labels placed on sample listings where one or more user-specified pathnames would actually be printed.
Vx.y	Is a generic label placed on sample listings where the version number of the product that produced the listing would actually be printed.
[ ]	Brackets indicate optional arguments or parameters.
{ }	One and only one of the enclosed entries must be selected unless the field is also surrounded by brackets, in which case it is optional.

{ }...	At least one of the enclosed items must be selected unless the field is also surrounded by brackets, in which case it is optional. The items may be used in any order unless otherwise noted.
|	The vertical bar separates options within brackets [ ] or braces { }.
...	Ellipses indicate that the preceding argument or parameter may be repeated.
[,...]	The preceding item may be repeated, but each repetition must be separated by a comma.
punctuation	Punctuation other than ellipses, braces, and brackets must be entered as shown. For example, the punctuation shown in the following command must be entered: `SUBMIT PLM86(PROGA,SRC,'9 SEPT 81')`
`input lines`	In interactive examples, user input lines are printed in white on black to differentiate them from system output.
`<cr>`	Indicates a carriage return.

CONTENTS

TABLES

FIGURES

CHAPTER 1 iNDX OVERVIEW

Introduction

This chapter provides an overview of iNDX (Intel Network Distributed Executive Operating System) under which the Series IV development system runs. iNDX provides you with dual execution modes (8080/8085 and 8086/8088) on the Series IV Development System. To execute under the 8085 mode you invoke ISIS-IV, a subsystem of the iNDX operating system. This chapter introduces you to the features of ISIS-IV, the 8080/8085 user interface on the Series IV.

This manual assumes you have read the *Intellec Series IV Microcomputer Development System Overview*, 121752, and the *Intellec Series IV Operating and Programming Guide*, 121753.

iNDX Operating System

The iNDX operating system provides you, as a Series IV user, with the following capabilities:

- File and device management
- Program execution control
- Library of system interfaces accessible via ISIS-IV

8086/8088 Execution Mode

The host execution mode of the Series IV is 8086/8088. Under this mode of iNDX you can perform iAPX86/88 operations. Refer to the *Intellec Series IV Operating and Programming Guide*, 121753, for information on 8086/8088 execution on a Series IV system.

8080/8085 Execution Mode

ISIS-IV is the subsystem of iNDX that provides you with the environment to execute 80/85 operations. ISIS-IV provides you with a convenient environment for source editing, assembly/compilation, linking, locating, debugging, and simulation.

Foreground/Background Mode

The iNDX operating system provides you, as the Series IV user, with the capacity of foreground/background processing. All jobs that are user-interactive are foreground jobs (e.g., editing or debugging). The foreground job is explicitly activated when you log on to the system; the job ends when you log off.

Background jobs are executed simultaneously with foreground jobs. They differ only in the fact that background jobs are not capable of user interaction. ISIS-IV will not take a background command, but will operate in the background mode (e.g., SUBMIT command).

Standalone/Network Mode

The iNDX operating system has been specifically designed to function similarly in a standalone and in a network mode. ISIS-IV, a subsystem of iNDX, executes in either mode of operation on the Series IV Development System.

The 8086/8088 host execution mode of iNDX provides the environment for performing network operations (e.g., importing and exporting jobs). Refer to the *Intellec Series IV Operating and Programming Guide*, 121753, for more details.

iNDX File Structure

The iNDX operating system employs a hierarchical file system that enables you to group your data logically. The base of the file system is the logical system root that connects the volumes within the file system. The root stores directory information for all the volumes. The root is the highest level directory in the distributed file system of iNDX. Each volume represents one physical mass storage device (e.g., a Winchester device).

Volumes are further divided into files. Files can be either directory files, which consist of directories and data, or data files, which consist of data only (see figure 1-1). Each volume can contain as many files (directory or data) as available storage will allow.

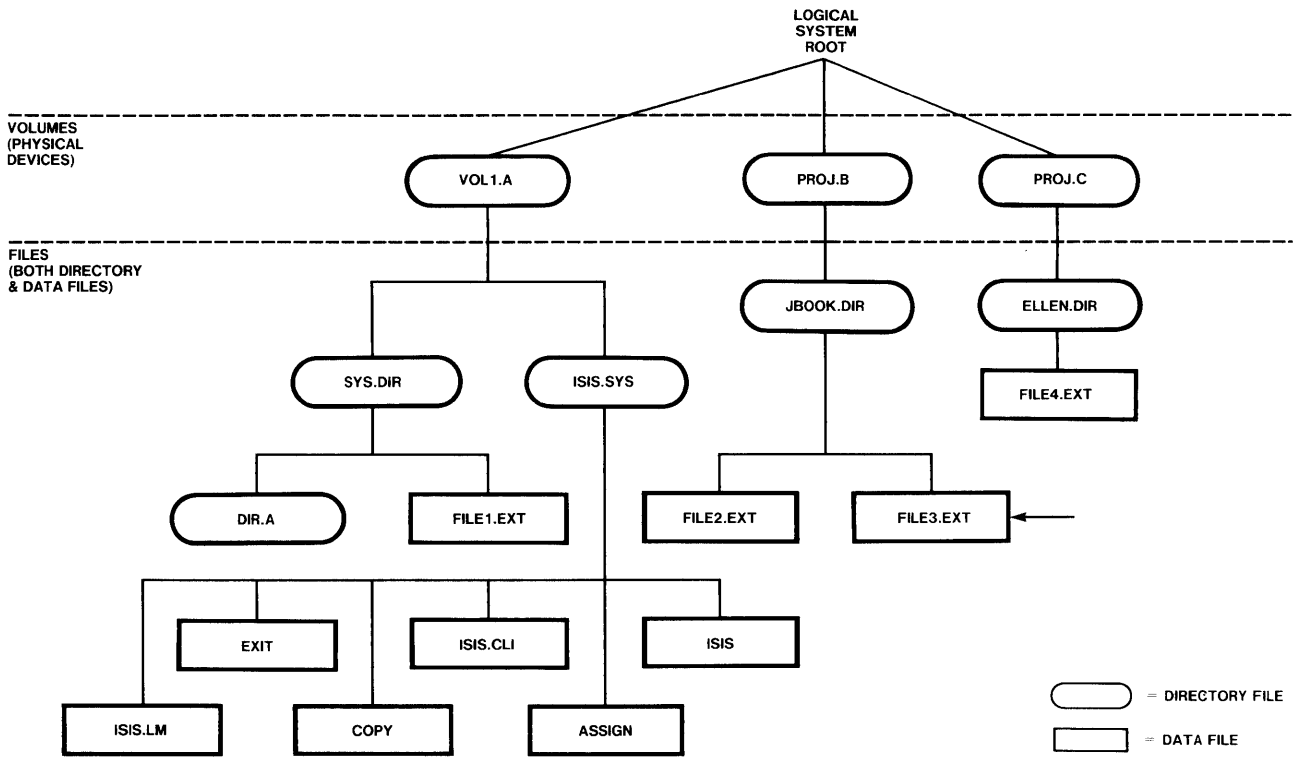


Figure 1-1. Sample iNDX File Structure

121880-1

The iNDX file system provides you with a file protection feature. For details on this file protection feature, iNDX file access rights, refer to Chapter 2.

ISIS System Directory

Basic ISIS-IV system files must be present for the ISIS program to be invoked. These basic ISIS-IV system files (e.g., ISIS, ISIS.LM, ISIS.CLI, and EXIT) are provided to you in a directory labeled ISIS.SYS. This directory also contains file maintenance programs (e.g., COPY, DELETE, DIR).

The ISIS-IV system directory, ISIS.SYS, can be within a volume root directory, or within another directory in the iNDX hierarchical file structure (see figure 1-1).

Entering ISIS-IV

You invoke ISIS-IV by executing the program ISIS. The ISIS program, found in the ISIS.SYS directory, loads in ISIS.LM and ISIS.CLI. The ISIS program must have access to these two files located in the ISIS system directory.

ISIS must have the logical name 0 defined for the ISIS system directory. You can use the Series IV LNAME command to make this definition. If the logical name 0 is not defined for its directory, ISIS will try to define the logical name for you. However, ISIS will look only in the volume root directory for the ISIS system directory.

Therefore, when ISIS.SYS directory is located in the volume root directory of the system, the ISIS program can be invoked by identifying it by its fully qualified pathname. For instance,

```
>/volume root /ISIS.SYS/ISIS <cr>
```

When ISIS-IV is invoked, ISIS.SYS becomes the system directory, :F0:. See the ASSIGN command in Chapter 2 for more information on directory identifiers.

The Series IV LNAME command enables you to access the ISIS program whether ISIS.SYS is located in the volume root directory or another directory. For instance,

```
>LNAME Define 0 For /parent directory/ISIS.SYS <cr>
>0/ISIS <cr>
```

This ISIS command may be given interactively or from a SUBMIT file. Refer to the *Intellec Series IV Operating and Programming Guide*, 121753, for information on the LOGON command.

Exiting ISIS-IV

You may exit ISIS-IV in two ways:

- Use the ISIS-IV EXIT command to return control to the host 8086/8088 mode of the iNDX operating system.
- Press the break key, located directly above the cursor control keys, and answer the displayed question when prompted. This method of exiting is useful if for some reason the EXIT program cannot be used—for instance, if it is accidently deleted.

Functions of ISIS-IV

iNDX File Access

One of the major functions of ISIS-IV is to enable you, as an ISIS user of a Series IV system, to access iNDX files. The ISIS ASSIGN command allows you to gain access to files in the distributed file system of iNDX. You access local (private) and remote (public) iNDX files with the same procedure. You do not have to know where the iNDX file is located physically. If you have the proper access rights to the file, you use the same commands to access a local file of a standalone system, a local file of a network workstation, or a remote file on a mass storage device of the network. Refer to the File Access section and the ASSIGN command in Chapter 2 for more details.

File Creation and Management

You can use ISIS-IV to create, revise, and manage files. These files can be either program files that are generated by an assembler or compiler, or files that are created through the use of the Text Editor. Refer to Chapter 2 for more details on file creation and management.

Text Editing

CREDIT is the screen-oriented editor that is supported by ISIS-IV on the Series IV system. CREDIT lets you display a file, move the cursor to any point in the text, make insertions, deletions, or other corrections, and see the results of the changes immediately. You can page forward or backward through the file and correct misspellings by simply positioning the cursor at the incorrect character and typing the correct one.

CREDIT also has a set of commands for command-mode editing. These include the more complex editing functions such as move, copy, command iterations, macro definition, and external file operations.

CREDIT also includes a Help command that displays the format and a brief functional description of each command. See the *ISIS-II CREDIT CRT-Based Text Editor User's Guide*, 9800902, for details on CREDIT.

Use of ISIS-IV by Other Programs

You can write your own Series IV programs that include ISIS-IV system calls. See Chapter 3 for more details.

MON 85

ISIS-IV also supports the ISIS Execution Unit Monitor (MON 85). MON 85 provides you with the basic utility functions for debugging 8080/8085-based programs by allowing you to

- Execute, single step, and breakpoint a program
- Display, modify, and scan memory
- Input from and output to I/O ports
- Perform arithmetic operations
- Disassemble instructions

CHAPTER 2
FILE CREATION AND MANAGEMENT

This chapter describes how to use ISIS-IV to create, revise, and manage files. In most cases the files will be user files generated through the use of an assembler or compiler. However, ISIS-IV can also create data files through the use of Text Editor commands that are designed to add, delete, and replace characters or lines as displayed on the system console.

This chapter begins with a description of the types of files that you will be manipulating. Next, the various aids that are available to you as a console user are explained. These aids are operating hints or simply descriptions of system characteristics that should be understood before attempting more complex tasks. The chapter then goes on to explain the use of ISIS-IV commands that allow you to create, revise, or delete files and/or references to files within a directory. CREDIT CRT-Based Text Editor is described in the *ISIS-II CREDIT CRT-Based Text Editor User's Guide*, 9800902.

File Characteristics

Naming, protecting, and accessing of files by an ISIS-IV user are described for you in the following sections.

iNDX File Structure

The iNDX file structure is a hierarchical (or inverted tree) file structure. Volumes that are established during the System Generation process can contain directory or data files. Each volume can contain as many files (directory or data) as available storage will allow. A directory file may contain other directory files or data files. Data files contain only data; a data file cannot contain a directory file (see figure 1-1).

Filenames

Every file is identified by a filename (e.g., FILE3.EXT) that can have two parts: a filename and an optional extension, separated by a period. The filename is a sequence of from one to six ASCII characters; an extension is a sequence of from one to three ASCII characters or a null extension with no separator—".".

You can use the extension of the filename to reflect the type of data in the file. It is important that you know the kind of data in a file so you can know what can be done with the file. For example, source programs could have the extension .SRC (or .ASM).

The object code produced by the translators could have the extension .OBJ; in fact, if you do not supply a name for the output file, the translators use the name of the input file with the extension .OBJ.

Many ISIS-IV programs in addition to the translators assign a specific extension on the input file if you do not supply it. One program, SUBMIT, assumes the extension .CSD on the input file, builds a .TMP output file, and then reassigns the console to the .TMP file. Other ISIS-IV programs create temporary working files with this .TMP extension (e.g., EDIT.TMP, LOCATE.TMP, LINK.TMP). This saves you time entering commands but can lead to misunderstandings if you are not aware of what the program does.

Filenames ending in .TMP that are created while you are operating in the 8080/8085 execution mode are mapped into Series IV (8086/8088 execution mode) filenames. This mapping of temporary filenames is performed to prevent SUBMIT, a compiler, or an editor from conflicting on temporary files when the programs are being executed on two network workstations.

Temporary files are renamed or deleted usually before the program that created the temporary files exits. However, if the program is aborted, the temporary file will remain. To enable another user at your workstation to re-use the same temporary filename without receiving access right violation messages, ISIS-IV automatically grants all users (WORLD) read, write, and delete access rights to the temporary file. See the following section on file protection for more information on access rights.

Wild Card Filenames

Some ISIS-IV commands allow you to specify filenames using a wild card construct. This means you can use an asterisk (*) or a question mark (?) to replace some or all of the characters in a name or extension (see the examples that follow).

1. The following special characters mean match anything when searching a directory for a filename:

*name.**	Match any filename with name and any extension or without an extension.
**.extension*	Match any filename with extension and any name.
.	Match any filename.

2. The asterisk can also specify a wild card match for the remainder of the name or extension but not for initial characters. For example,

   AB*HEX means match any filename with AB as the first two characters of the name and HEX as the extension. This example would match ABC.HEX, ABXYZ.HEX, and AB.HEX.

   *B.HEX is illegal since * must follow initial alphanumeric characters.

   *.BAK means match any filename with a .BAK extension. This example would match A.BAK, AB.BAK, or ABC.BAK.

3. The question mark specifies a single character for a wild card match. For example,

   A?B.HEX means match any filename with A and B as the first and third characters of a three-character name and HEX as the extension. This example would match ACB.HEX, AXB.HEX, and AMB.HEX.

   A??.* means match any filename with A as the first character of a three-character name and any extension.

4. :device: cannot include a wild card character.

File Protection

iNDX provides file access rights to protect your files from accidental addressing and destruction. All files in the hierarchical file structure have access rights that can be controlled with the ACCESS command. Data file access rights are read, write, and delete a file. Directory file access rights are list, add, and delete a directory.

The ATTRIB command allows you to control the write-protect status of a file. The ISIS-IV ATTRIB command maps the write-protect attribute of a file into the write or add access rights of iNDX. See the ATTRIB command in this chapter for further details.

File Access

The ISIS-IV ASSIGN command enables you to access both local and remote iNDX files when executing in the 8080/8085 mode. You do not have direct access as an ISIS-IV user to iNDX files. To access an iNDX file you must identify it by its fully qualified pathname. To identify a target data file by its fully qualified pathname, you must identify every volume and directory from the logical system root to the data file. For example, the fully qualified pathname for the data file FILE3.EXT is /PROJ.B/JBOOK.DIR/FILE3.EXT. The slash (/) acts as a delimiter between the names of the volume and the directories in the path along the "branches" of the "tree."

The ISIS-IV ASSIGN command allows you to use directory identifiers to represent the pathnames. For example, to assign a directory identifier to the fully qualified pathname of the directory that contains the data file FILE3.EXT you would type

```
-ASSIGN :F1: TO /PROJ.B/JBOOK.DIR <cr>
```

You can now access this data file FILE3.EXT by identifying it by its iNDX pathname —:F1:FILE3.EXT.

These directory identifier assignments that you create while operating under the ISIS-IV mode are not erased if you transfer to the host mode (8086/8088) of Series IV. However, logging off will delete the directory identifier assignments.

Console User Aids

In manipulating files, as an ISIS-IV user, various console aids are available to you.

Line Editing

Prior to line termination, you can revise or delete the buffer contents by using special non-printable editing characters and character combinations. These editing characters are not normally stored in the line-editing buffer but rather provide control over the buffer contents. The editing characters are

RUBOUT	Deletes the preceding character. Repeated usage is allowed. (On systems using a teletypewriter, RUBOUT echoes the deleted character to the teletypewriter.)
CONTROL-P	Used before another editing character (including itself) to allow entry of the editing character into the line editing buffer.
CONTROL-R	Displays the current line entered.
CONTROL-X	Deletes the entire contents of the line entered, and displays a number sign (#) followed by a carriage return and a line feed.
CONTROL-Z	Enters an end of file, deletes the contents of the line editing buffer, and returns 0 bytes of data to the caller.

NOTE

The following characters are not editing characters, but they affect terminal output:

CONTROL-S	Suspends terminal output and delays program execution.
CONTROL-Q	Resumes terminal output after the CONTROL-S command is given.

The line-editing facility can be applied to files other than the console input device. There is a complete description of line-editing in the Line-Editied Input Files section, Chapter 3.

The System Console

The console, whatever device it is assigned to, is always the source of system commands. The SUBMIT command directs ISIS-IV to take commands from a file. The SUBMIT file can return control to the initial system console by means of a CONTROL-E. The console can also use CONTROL-E to return control to the SUBMIT file.

When CONTROL-E is input to ISIS-IV as part of the command line either from the SUBMIT file or the console keyboard, CONTROL-E is echoed but is not entered in the input buffer. To enter a CONTROL-E into the input buffer and subsequently into a SUBMIT file, the CONTROL-E must be preceded by CONTROL-P, in which case CONTROL-E is entered as a literal.

Under ISIS-IV the files :CI: and :CO: are pseudonyms for the devices serving as console input and output. The :CI: file is always the source of system commands. The :CO: file receives console output such as the echo of a command. These two files are always open. However, it is not an error for a program to issue an OPEN system call for either of these files. Neither :CI: or :CO: count as one of the six files allowed open simultaneously by ISIS-IV.

When ISIS-IV is invoked, it makes assignments for :CI: and :CO:. The device that is assigned as console input and console output by ISIS is dependent upon the type of invocation.

It is impossible for an ISIS program to read from the keyboard (:VI:), or to print to the screen (:VO:) when it is being executed under background or import mode.

You may change the device assignments for console input and console output by using

- The CONSOL system call
- Your own program
- The SUBMIT program
- The Enhanced Command Line Interpreter (ECLI)
- The CONSOL tool box program

Using ISIS-IV Commands

The file management capabilities of ISIS-IV can be controlled in several ways:

- Direct entry of ISIS-IV commands at a console keyboard
- Issuance of ISIS-IV system calls by a program
- Entry of console commands into a file by using the Text Editor. Execution of this file via the SUBMIT command causes ISIS-IV to respond as if it were receiving commands directly from you. The advantages are that you need not be present when the submitted job is performed, and you need not re-enter the commands each time the job is submitted.

The following text defines only those commands input by you at the console. The definitions include a summary of the preparation of the file created by the command. The use of system calls is described in Chapter 3.

ISIS-IV console commands perform three basic tasks:

- Executes programs
- Creates, deletes, and revises files and directories
- Converts object file formats

The ISIS-IV console commands associated with each of the preceding tasks is identified in the following five sections.

Program Execution

filename	Execute the program named *filename.*
DEBUG	Load a program and give control to the Monitor.
EXIT	Terminate ISIS-IV execution and return control to the host execution mode (8086/8088).
SUBMIT	Create a file to act as console input.

File Maintenance

ACCESS	List or change the iNDX access rights of a file.
ASSIGN	Assign a directory identifier to a directory pathname.
ATTRIB	List or change the write-protect status of a file.
COPY	Copy a file from one directory to another.
CREATE	Create a new directory in the iNDX file structure.
DELETE	Remove references to a file from the directory and free storage space associated with that file.
DIR	Output the names of and information about the files listed within the directory.
REMOVE	Delete an empty directory.
RENAME	Change the name of a file.
SPACE	Display volume information of the specified volume.
VERS	Display ISIS utility program version numbers.
WHO	Display the name of the user who is currently logged on.

File Editing

CREDIT	Create and modify files.

The text editor subcommands are described in the *ISIS-II CREDIT (CRT-Based Editor) User's Guide*, 9800902.

Code Conversion

HEXOBJ	Convert a program from hexadecimal to object module format.
OBJHEX	Convert a program from object module to hexadecimal format.

Program Control

LIB	Create and control program libraries.
LINK	Combine program files and resolve external addressing.
LOCATE	Convert relocatable object to absolute addresses for execution.

Refer to the *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems*, 121617, for information on program control commands.

Command Syntax

The general syntax of an ISIS-IV console command is

command parameters< c r >

where

command	is the name of a program.
parameters	are one or more items required by the command. When entering more than one parameter, separate them with command or blank spaces unless otherwise noted under the individual commands. When a parameter consists of switches, separate them by spaces, not by commas.

In most cases, a command executes when the carriage return is encountered. Any exceptions are noted under the individual commands.

NOTE

Some of the ISIS-IV commands (8080/8085 mode) have the same name and a similar function as the Series IV commands (8086/8088 mode). However, the syntax of the commands is different. You must use the correct command syntax for the corresponding execution mode.

Program Execution Commands

You can call a program for direct execution, in which case you must respond to any queries from the program and to any errors encountered during program execution. You can also call for execution of the program under the Monitor, in which case the debugging provisions of the Monitor aid you in identifying and locating program errors. Finally, you can submit the program as a job to be handled by the system without any interaction on your part. In this latter case you must prepare a file that interacts with the program in the same manner as you would during direct execution of the program.

Filename — Direct Program Execution

Because ISIS-IV commands (except DEBUG) are actually the names of files containing executable programs, simply enter the name of the file for program execution. You may also include parameters with the filename to provide control over the program to be executed. However, the program must be written to accept these parameters and must read the parameters from the line-editing buffer. Refer to Chapter 3 for more information on the line editing of command lines.

DEBUG — Execute a Program Under the Monitor

Syntax

```
DEBUG [[:Fn:]filename] <cr>
```

where

:F*n*:	is the directory identifier of the directory where the file, *filename*, resides. The value *n* is an integer between 0 and 9 inclusive. If :F*n*: is not specified, :F0: is the default.
filename	is any ISIS-IV command file or the filename of any executable program. The program must be an absolute object module. If *filename* is omitted, control transfers to the Monitor, but no program is loaded.

Description

When your executable 8080/8085 program is loaded, the Monitor displays the contents of the program counter and prompts for a command with a period (.) on the system console.

Begin execution of the program by entering the Monitor G command. You may specify a starting address (entry point address) and up to four breakpoint addresses in the G command.

When execution of your program is suspended at the breakpoint address, use other Monitor commands to inspect and/or change the contents of memory and/or registers. The Monitor N command will single step your program. Continue program execution from the point of suspension with another G command.

You can return to ISIS-IV from the debug mode and reset the debug switch by entering the Monitor G8 command.

Examples

1. This example shows a successful attempt to execute a program named LIST in debug mode at a load address of 37E1:

```
-DEBUG LIST FILE.TXT<cr>
*BREAK* at 37E1
.G<cr>
```

2. This example executes the same program in debug mode, suspends execution at the specified breakpoint address, and then returns to ISIS-IV with a G8 command instead of letting the program issue an EXIT system call:

```
-DEBUG LIST FILE.TXT<cr>
*BREAK* at 37E1
.G, 36A0<cr>
```

(Use Monitor commands to examine registers and memory when the breakpoint 37E1 is reached.)

```
.G8<cr>
iNDX-S41 (Vx.y) ISIS-IV Vx.y
```

3. This example transfers control to the Monitor with no program loaded. Enter the Monitor G command with no address to return control to ISIS-IV.

```
-DEBUG<cr>
#0008
```

EXIT — Terminate ISIS-IV Execution

Syntax

```
EXIT <cr>
```

Description

The EXIT command terminates ISIS-IV execution. Control transfers from the 8080/8085 mode of ISIS to the 8086/8088 host execution mode of Series IV. The appearance of the iNDX prompt (>) indicates that control has been returned to the host execution mode.

NOTE

You may also terminate ISIS-IV execution by pressing the break key and then selecting the appropriate displayed option.

All of the directory identifier assignments that were made in ISIS will remain after the EXIT command is executed.

SUBMIT — Take Console Commands from a Disk File

Syntax

```
SUBMIT [:Fn:]filename[(parameter[, . . .])]<cr>
```

where

:F*n*:	is the directory identifier of the directory or drive where the file resides.
filename	is the name (and extension, if any) of the file that contains the command sequence definition. If the extension is omitted, SUBMIT assumes the default extension .CSD.
parameter	is an actual value that is to replace a formal parameter in the command sequence definition file. The maximum number of parameters allowed is ten. If you omit a parameter from the SUBMIT list, enter a comma in its place. Null parameters are allowed.

A parameter is a character string of up to 31 characters. Any ASCII character from 20H to 7AH is legal, except a comma, space, or right parenthesis. If a parameter contains a comma, space, or right parenthesis, enclose the parameter in apostrophes. To use an apostrophe inside a parameter with an apostrophe, use two apostrophes in its place. For example,

```
'TITLE(''QUOTE ('') SEARCH ROUTINE'')'
```

is used in the final command as

```
TITLE('QUOTE (') SEARCH ROUTINE')
```

Description

The SUBMIT command causes ISIS-IV to take its commands from a file rather than from the console.

SUBMIT uses two files:

- A command sequence definition (CSD) file that contains the command sequence definition. You create this file with formal parameters, using the editor.
- A temporary .CSD file that contains the command sequence to be executed.

  SUBMIT creates this file with the actual parameters supplied in the SUBMIT command that replaces the formal parameters. The temporary file has the same name as the command sequence definition file but with the extension .TMP. Do not modify this file.

  SUBMIT reassigns the console input device to the .TMP file it creates and returns control to ISIS-IV, which then executes the commands in the .TMP file. The .TMP file has a final command that restores the console input device to its former device assignment and deletes the .TMP file.

When creating the CSD file, specify formal parameters by using two characters, %*n*, where *n* is a digit from 0 through 9. You may place formal parameters anywhere in the CSD file. To enter a percent sign (%) that is not to be interpreted as a formal parameter, enclose it in single quotes.

Any program—except a LOGON, LOGOFF, or IMPORT (ASSIGN cannot be used in a nested SUBMIT file)—that reads its commands from :CI: noninteractively can be executed.

The CSD file can also contain commands to the programs being run. Using a SUBMIT command in a CSD file causes another .TMP file to be created. SUBMIT commands can be nested to any depth.

A CNTL-E (↑E) in a .TMP file switches the console input from the .TMP file to the initial system console, allowing interactive processing. To return control to the .TMP file, enter CNTL-E at the console. If control is *not* returned to the .TMP file, or if an error occurs after a command sequence has started processing, control returns to ISIS-IV and the .TMP file is not deleted.

Any program running under SUBMIT must allow two buffers in addition to the open files and buffers required by the program itself. See the *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems*, 121617, for information on how to determine the base address of your program.

Possible Error Conditions

Do not use LOGON, LOGOFF, ASSIGN, or IMPORT commands in a SUBMIT file. Including any of these commands in the file causes an error message at the workstation.

```
ILLEGAL LOGON WHILE :CI:/:CO: FILE ON NETWORK
```

or

```
ILLEGAL LOGOFF WHILE :CI:/:CO: FILE ON NETWORK
```

Examples

1. This example shows a PL/M-80 compilation, a LINK, and a LOCATE executed from a SUBMIT file with two directories. A CNTL-E is entered in the command sequence definition after the PL/M compilation so you can remove the compiler disk. When the regular system disk (with LINK and LOCATE) is mounted, you enter CNTL-E to resume processing.

   The file CMPLNK.CSD in drive 1 contains the following command sequence definition. See the *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems* (121617) for an explanation of controls in the PL/M-80 command. The CMPLNK.CSD file contains

   ```
   PLM80 %0.%1 DEBUG XREF DATE(%2)
   ↑E
   LINK %0.OBJ,SYSTEM.LIB TO %0.SAT&
   PRINT(%0.MP1) MAP
   LOCATE %0.SAT PRINT(%0.MP2) MAP
   ```

The SUBMIT command entered to compile, link, and locate PROGA.SRC is:

```
-SUBMIT :F1:CMPLNK (:F1:PROGA,SRC, '3 OCT 81')<cr>
```

The command sequence actually executed is shown as it would be echoed on the console output device:

```
-PLM80 :F1:PROGA.SRC DEBUG XREF DATE(3 OCT 81)

ISIS-IV PL/M-80 COMPILER V3.1
PL/M-80 COMPILATION COMPLETE 0 PROGRAM ERROR(S)

-↑E↑E
-LINK :F1:PROGA.OBJ,SYSTEM.LIB TO :F1:PROGA.SAT &
**PRINT(:F1:PROGA.MP1) MAP
-LOCATE :F1:PROGA.SAT PRINT(:F1:PROGA.MP2) MAP
-:F0:SUBMIT RESTORE :F1:CMPLNK.TMP(:VI:)
-
```

File Control Commands

The file control commands are those commands that work directly with files and are not concerned with the type of information within the files. This is in contrast to the other ISIS-IV console commands that deal with program files specifically.

The file control commands provide for copying or deletion of files and revision or display of file directory contents.

ACCESS — List or Change Access Rights of a File

Syntax

$$\texttt{ACCESS}\left\{\begin{array}{l}\texttt{:F}n\texttt{:}\\ pathname/\end{array}\right\}\ filename\,[switch]\texttt{<cr>}$$

where

:F*n*:	is the directory identifier that contains *filename*. The value *n* is an integer between 0 and 9 inclusive. If :F*n*: is not specified, :F0: is assumed.
pathname	is a fully qualified pathname.
filename	is the name and extension, if any, of the file.
switch	for the OWNER ACCESS RIGHT consists of three parts: the OWNER Identifier, the ACCESS Identifier and the RIGHT Identifier (see table 2-1).

Description

The ACCESS command lists or changes the Owner or World access rights of a data or directory file. These switches can be used to protect files from accidental change or deletion.

Listing Current Access Rights. To list the current access rights of a file, type

```
-ACCESS :F2:filename.ext<cr>
```

Changing Access Rights. To change the access rights of a data file or a directory file, use the ACCESS command switches. The 24 different switches consist of three characters typed with no intervening space.

The first character indicates the Owner Rights to be changed—the file owner (Owner) or the other users (World).

The second character indicates the type of access to be altered. A file has three access rights: Read, Write, and Delete. A directory also has three access rights: List, Addentry, and Delete.

The third character indicates the access right as being denied (reset) or granted (set). A zero (0) indicates the switch is off and the access right is denied. A one (1) indicates the switch is on and the access right is granted.

After the access rights of a file or a directory are changed, the new access rights are displayed.

Table 2-1. iNDX Access Rights

Identifier	Options
OWNER	O — Owner of the file or directory W — World or public users
ACCESS	Data files: R — Read a file W — Write a file D — Delete a file Directory files: L — List a directory A — Add a directory entry D — Delete a directory
RIGHT	0 — Deny access right (reset, off) 1 — Grant access right (set, on)

To allow the World to read a file, set the World Read switch ON.

```
-ACCESS FILE1.EXT WR1<cr>
```

To allow the public to add a directory, set the World Add switch ON.

```
-ACCESS MYDIR.ALL WA1<cr>
```

Altering Access Rights of Multiple Files and Directories. The access rights of more than one file or directory can be changed simultaneously. However, the access rights of the files must be changed to a common switch. To "turn off" the Owner Delete rights to files FILE1.EST, FILE2.EXT, and FILE3.EXT, type

```
-ACCESS FILE*.EXT OD0<cr>
```

Possible Error Conditions

An error occurs when

- You try to access a file or a directory that does not exist.
- You do not have access rights to change the access characteristics of a file or a directory.

Examples

1. This example denies the World Add access rights of a file.

```
-ACCESS NEWFIL.EXT WA0<cr>
```

2. This example grants the Owner Read and Write access rights to any file that matches in an ISIS wild card search for DAT*.*.

```
-ACCESS DAT*.* OR1 OW1<cr>
```

ASSIGN — Assign a Directory Identifier

Syntax

```
ASSIGN [{n   }TO y] <cr>
        {:Fn:}
```

where

n	is a number 0 to 9. *n* and :F*n*: are directory identifiers. The directory identifier for the default system drive is :F0:. The directory :F0: must contain the files ISIS, ISIS.LM, ISIS.CLI, and EXIT.
y	is one of the following: • A fully qualified directory pathname • The word NULL

Description

The ASSIGN command allows a directory identifier to be mapped into the iNDX hierarchical file system. ASSIGN enables the ISIS-IV user to access iNDX files with ISIS file naming conventions. A full understanding of the difference between iNDX and ISIS file naming conventions is necessary. Refer to the *Intellec Series IV Operating and Programming Guide*, 121753, for information on Series IV iNDX file naming conventions.

Listing Current Assignments. To list current assignments of directory identifiers to hierarchical directories type

```
-ASSIGN <cr>
```

This ASSIGN command generates a listing of all directory identifiers and their current assignments (see the following example).

```
DEVICE    ASSIGNED TO

:F0:      /VOL1.A/ISIS.SYS
:F1:      /PROJ.B/JBOOK.DIR
:F2:      /VOL1.A/SYS.DIR
:F3:      /VOL1.A/ISIS.SYS/FINISHED.MODS
:F4:      /PROJ.C
:F5:      /PROJ.C/ELLEN.DIR
:F6:      NULL
:F7:      NULL
:F8:      NULL
:F9:      NULL
```

In the preceding example VOL1.A, PROJ.B, and PROJ.C are volume root directories. The ISIS.SYS and SYS.DIR directories are subdirectories of the volume root directory VOL1.A. The JBOOK.DIR directory is a subdirectory of the volume root directory PROJ.B, ELLEN.DIR is a subdirectory of PROJ.C, and FINISHED.MOD is a subdirectory of ISIS.SYS.

Given the previously-listed assignments, the volume root directories VOL1.A and PROJ.B are not accessible because they have not yet been defined. The volume root directory PROJ.C is accessible because :F4: has been currently assigned to it.

Changing Directory Identifiers. Use the ASSIGN command with the previously-mentioned syntax conventions to change the directory identifier assignment(s). For example, to change the directory identifier assignment of :F1: from the directory /VOL1.A/ISIS.SYS to the directory /PROJ.C, type

```
-ASSIGN :F1: TO /PROJ.C <cr>
```

or

```
-ASSIGN 1 TO /PROJ.C <cr>
```

NOTE

The directory identifier :F0: should always be assigned to the ISIS system directory. You can reassign :F0: as long as the new directory has all of the necessary ISIS system files (e.g., ISIS, ISIS.LM, ISIS.CLI, and EXIT).

You must always use a fully qualified pathname when making directory identifier assignments.

Defining New Directories. If you need a new directory, use the CREATE command, and then define this new directory with the ASSIGN command. For example,

```
-CREATE /VOL1.A/ISIS.SYS/NEW.DIR <cr>
-ASSIGN 1 TO /VOL1.A/ISIS.SYS/NEW.DIR <cr>
```

Now you can use the directory identifier :F1: to access the NEW.DIR directory.

Examples

1. In the following example the directory identifier :F3: is assigned to :F3:.

```
-ASSIGN 3 TO /VOL1.A/ISIS.SYS <cr>
```

   Now any reference to :F3: will be the same as referencing :F0:.

2. Use the ASSIGN command to list the current assignments. For example,

```
-ASSIGN <cr>

DEVICE   ASSIGNED TO

:F0:     /VOL1.A/ISIS.SYS
:F1:     /PROJ.B/JBOOK.DIR
:F2:     /VOL1.A/SYS.DIR
:F3:     /VOL1.A/ISIS.SYS
:F4:     /PROJ.C
:F5:     /PROJ.C/ELLEN.DIR
:F6:     NULL
:F7:     NULL
:F8:     NULL
:F9:     NULL
```

3. The following examples an illegal use of an unqualified pathname.

```
-ASSIGN 3 TO 0 <cr>
```

   or

```
-ASSIGN 3 TO 0/NEW.DIR <cr>
```

The 0 represents :F0: when executing in the 8080/8085 mode, and not /VOL1.A/ ISIS.DIR as it would when executing in the host 8086/8088 mode of the iNDX operating system.

NOTE

The directory identifier assignments that you make, as an ISIS-IV user, are not deleted when you transfer to the 8086/8088 execution mode. (The assignments remain after the EXIT command is executed.) However, logging off will delete the directory identifier assignments.

ATTRIB — List or Change the Write-Protect Status of a File

Syntax

```
ATTRIB [:Fn:]filename [attriblist] [Q]<cr>
```

where

:F*n*:	is the directory identifier of the directory where the file resides.
filename	is a file whose write-protect status is to be changed or displayed. The wild card construction can be used to change and/or display the write-protect status of a group of files.
attriblist	is one or more of the following:
W0 or W1	Resets (W0) or sets (W1) the write-protect attribute of a local or remote file. When set, the file cannot be opened for output or update, and cannot be deleted or renamed.
Q	Specifies query mode operation.

If two values of the write-protect attribute are specified—for example, both W0 and W1—the one rightmost in the command takes precedence.

Description

The ATTRIB command changes and/or displays the write-protect status of a local or a remote file. ATTRIB allows you to change the write-protect status of a file by mapping this attribute into the iNDX access rights (see table 2-2).

When you specify the Q switch, ATTRIB displays the following messages before changing the write-protect status of a file.

```
filename, MODIFY ATTRIBUTE?
```

Type a Y or y to modify the write-protect status of the file. Any other response causes ATTRIB to leave the write-protect attribute unchanged for the specified file and to go on to the next file in the group. When attribute for a file has been changed, the current write-protect status for the file is displayed.

If a nonexistent file is specified, ATTRIB displays

```
filename, NO SUCH FILE
```

If a non-disk file is specified, ATTRIB displays

```
filename, NON-DISK DEVICE
```

Possible Error Conditions

An error occurs when

- The file does not exist.
- The directories are not assigned.

Table 2-2. Write-Protect Status of a File

Operation	Action
W1 — Set write-protect on local files	Deny WORLD WRITE Deny WORLD DELETE
W1 — Set write-protect on remote files	Deny WORLD WRITE Deny WORLD DELETE Deny OWNER WRITE Deny OWNER DELETE
W0 — Reset write-protect on local files	Grant WORLD WRITE Grant WORLD DELETE
W0 — Reset write-protect on remote files	Grant OWNER WRITE Grant OWNER DELETE

Examples

1. This example changes the write-protect attribute of a group of files.

```
-ATTRIB PROGA.* W1<cr>
   FILE              CURRENT ATTRIBUTES
 :F0:PROGA.SCR                W
 :F0:PROGA.OBJ                W
```

COPY — Copy a File

Syntax

```
COPY [:Fn:]infile [,...] TO {[:Fn:][outfile]} [switches] <cr>
                            {:device:       }
```

where

:F*n*:	refers to the directory identifier of the directory where the file resides.
infile	is a file (or group of files when using the wild card construct) to be copied. The copy does not affect the contents of *infile*. If more than one *infile* is specified, they are concatenated in the order specified. When concatenating files, specify the full name and extension of each file; do not use the wild card construct.
outfile	is a file to be created or recreated. If :F*n*: is not specified, :F0: is assumed. *outfile* must include the extension, if any. If *outfile* is not specified, :F*n*: must be specified.
:*device*:	is an output device, such as :LP:, :SP:, or :CO:.
switches	are one or more of the following:
Q	Specifies the query mode. The system displays the following message before a copy is performed—COPY *infile* TO *outfile* ?. A yes or y response causes the copy to be performed. Any other response causes the copy not to be performed.
B	Deletes an existing file without displaying the ALREADY EXISTS prompt. The existing file is deleted and recreated with new data.
U	Opens *outfile* for update instead of deleting it. The ALREADY EXISTS message is suppressed. The length is not changed unless the copy causes an increase in the size of the file.
	If U and B are both specified, the U function is performed.

Description

The COPY command copies files from one directory to another.

When copying from one directory to another, the destination can be directory files or physical devices. The copy must be made from an input device to an output device. For example, you can copy from the console to the printer but not from the printer to the console.

You must have read access rights to the infile (source file). If outfile (destination file) is an existing file that you have been granted delete access rights to, the following message is displayed:

```
outfile FILE ALREADY EXISTS
DELETE?
```

A yes or y response (followed by a carriage return) causes COPY to delete the existing file before making the copy. No change is made for any other response.

If you have been denied delete access rights to *outfile*, the following message is output:

```
outfile iNDX FILE ACCESS RIGHTS VIOLATION
```

Wild Card Designations. When you use wild card designations, the following rules apply:

- Every position in the *infile* name that contains an asterisk must have a corresponding asterisk in the *outfile* name.
- Every position in the *infile* name that contains a question mark must have a corresponding question mark or asterisk in the *outfile* name.
- The wild card characters cannot be used in directory designations (you cannot specify :F*:).
- You must have list access to the source directory.
- You must have add-entry access to the destination directory.

To copy files selectively with the wild card construct, use the query mode, for example,

```
-COPY :F0:CHAP?.DFT TO :F1: Q<cr>
```

The system then displays the query message before copying each file.

Copying to Another Directory. The COPY command provides a special case for convenience when copying directory files to a different directory.If *outfile* is to have the same name as *infile*, a specific *outfile* is unnecessary. For example,

```
-COPY :F1:ABC.XYZ TO :F2:<cr>
```

is the same as specifying

```
-COPY :F1:ABC.XYZ TO :F2:ABC.XYZ<cr>
```

This form can be used with wild card designations in *infile*.

```
-COPY :F1:*.*TO :F2:<cr>
```

At the end of the listing of files that were copied, the following message is displayed if you have been denied delete access rights to the specified files:

```
iNDX FILE ACCESS RIGHTS VIOLATION
```

The specified files to which you have been denied delete access rights are not copied.

Using a wild card designation when concatenating files causes an error message to be displayed.

```
-COPY A, BC.*TO D<cr>
WILD CARD DELIMITERS DURING CONCATENATE
```

When you use the concatenate operation, *outfile* must not have the same name as *infile*. If it does, the following error message results:

```
-COPY A, B TO B<cr>
SOURCE FILE EQUALS OUTPUT FILE ERROR
```

If the rules governing wild card designations are not followed, the following error message is displayed:

```
-COPY ABC.*TO D<cr>
FILE MASK ERROR
```

Possible Error Conditions

An error occurs when

- The source file is not present.
- The destination file already exists.
- You do not have read access rights to the source file (infile).
- You do not have delete access rights to the destination file (outfile).
- You do not have list access rights to the source directory when you used a wild card designation.
- You do not have add entry access rights to the designation directory when you used a wild card designation.

Examples

1. This example copies three files to one, overwriting its contents.

```
-COPY CHAP1,CHAP2,CHAP3 TO BOOK<cr>
:F0:BOOK FILE ALREADY EXISTS,
DELETE? Y<cr>
APPENDED :F0:CHAP1 TO :F0:BOOK
APPENDED :F0:CHAP2 TO :F0:BOOK
APPENDED :F0:CHAP3 TO :F0:BOOK
-
```

2. Example 1 could have been done in the following way:

```
-COPY CHAP1,CHAP2,CHAP3 TO BOOK U<cr>
APPENDED :F0:CHAP1 TO :F0:BOOK
APPENDED :F0:CHAP2 TO :F0:BOOK
APPENDED :F0:CHAP3 TO :F0:BOOK
-
```

3. This example lists a file on the line printer.

```
-COPY BOOK TO :LP:<cr>
COPIED :F0:BOOK TO :LP:
```

4. This example copies a file from directory 0 to directory 1.

```
-COPY PROGA TO :F1:NEWPRG B<cr>
COPIED :F0:PROGA TO :F1:NEWPRG
-
```

5. These examples show valid uses of wild card names with the COPY command.

```
-COPY :F1:*.* TO :F2:<cr>

-COPY :F1:A??C TO :F0:D??E<cr>

-COPY :F1:A?????? TO :F0:B*.CPY<cr>
```

6. This example shows a valid use of the COPY command concatenating files on a directory.

```
-COPY :F6:REMTE.1,:F6:REMTE.2 TO :F6:REMTE.3<cr>
APPENDED :F6:REMTE.1 TO :F6:REMTE.3
APPENDED :F6:REMTE.2 TO :F6:REMTE.3
-
```

7. When the workstation is connected to the network, this example prints the file :F1:BOOK at the network spooled printer.

```
-COPY :F1:BOOK TO :SP:<cr>
```

8. When the workstation is connected to the network, this example copies a file, :F3:MYFILE.TXT, to the network spooled printer.

```
-COPY :F3:MYFILE.TXT TO :SP:<cr>
:F3:MYFILE.TXT COPIED TO :SP:
```

CREATE — Create a Directory

Syntax

```
CREATE { :Fn:      } new directory name<cr>
       { pathname/ }
```

where

pathname	is an existing directory.
:F*n*:	is the directory identifier assigned to a fully qualified directory pathname.
new directory name	is a string of up to 14 alphanumeric characters.

Description

The CREATE command creates a new directory in the iNDX file system. When specifying a directory identifier, the user may specify a fully qualified pathname or a pathname component prefixed by an assigned directory identifier.

All of the components in a pathname must point to existing directories, except the new directory name. The new directory name is the only component that specifies a non-existent directory.

Table 2-3 lists the default access rights of a new directory.

Any user (World) can use ACCESS to grant and to deny the World access rights of a directory on a 5¼ inch flexible diskette. However, Owner access rights to such a directory can not be granted or denied because there are no owners of directories on a 5¼ inch flexible diskette.

Only the owner can grant and deny Owner and World access rights to his directory on a Winchester device.

Table 2-3. Directory Access Rights

Operation	Action
CREATE a directory on a 5¼ inch flexible diskette	GRANT: World Delete Access World List Access World Add Entry Access
CREATE a directory on a Winchester device or hard disk	GRANT: Owner Delete Access Owner List Access Owner Add Entry Access DENY: World Delete Access World List Access World Add Entry Access

Possible Error Conditions

An error will occur when

- You have not logged on.
- You have used invalid syntax.
- You have not been granted add access rights to the parent directory.

- The parent directory does not exist.
- The new directory name already exists.
- The new directory name exceeds 14 characters.
- The directory identifier is not assigned.

NOTE

Use the REMOVE command to eliminate an unwanted, empty directory created with the CREATE command.

Examples

1. This example creates a directory in the /VOL/NRM1/COMPILERS directory.

```
-CREATE /VOL/NRM1/COMPILERS/SCRS<cr>
```

2. This example creates a directory identical to the directory in example 1.

```
-ASSIGN :F5: TO /VOL/NRM1/COMPILER<cr>
-CREATE :F5:SRCS<cr>
```

DELETE — Delete a File

Syntax

```
DELETE [:Fn:]filename [Q] [, . . . [Q]] <cr>
```

where

:F*n*:	refers to the directory identifier of the directory where the file resides.
filename	is the name of a file to be deleted. The wild card construction can be used to delete a group of files.
Q	Specifies the query mode. The system displays the following message before each file is deleted—*filename*, DELETE. A yes or y response causes the deletion. Any other response causes the deletion not to be performed.

Description

The DELETE command deletes specified directory entries if you have been granted delete access rights.

This command removes the specified file, directory, or group of files from a directory, making the space it occupied available for reassignment. A file to which you have not been granted delete access rights cannot be deleted.

If *filename* is a file to which you have delete access rights, the file is deleted and a confirming message is sent to the console.

If *filename* specified in the DELETE command does not exist, the following message is sent to the console:

```
filename, NO SUCH FILE
```

If the file cannot be deleted because you have not been granted delete access rights, the following message is sent to the console:

```
filename, iNDX FILE ACCESS RIGHTS VIOLATION
```

NOTE

Before a directory can be deleted, all of the directory entries must be deleted.

Query Mode. Using the Q switch displays the query message before deleting each file.

The query mode allows you to delete files selectively when using the wild card construct, for example,

```
-DELETE :Fn:CHAP?.* Q<cr>
```

The system then displays the query message for each file that matches the wild card construct.

Examples

1. This example deletes three files.

```
-DELETE CHAP?.*<cr>
:F0:CHAP1.TXT, DELETED
:F0:CHAP2.LST, DELETED
:F0:CHAP3.SRC, DELETED
-
```

2. This example shows an attempt to delete a file to which the user has not been granted delete access rights.

```
-DELETE PROGA.ASM<cr>
:F0:PROGA.ASM, iNDX FILE ACCESS RIGHTS VIOLATION
-
```

3. This example deletes a file from the spooled print queue when the workstation is connected to the network.

```
-DELETE :SP:BOOK<cr>
:SP:BOOK, DELETED
-
```

DIR — List a Directory

Syntax

```
DIR [FOR filename] [TO listfile] [switches] <cr>
```

The positions of these fields are not fixed.

where

filename	is the file (or group of files specified with the wild card construction) whose directory entry is to be listed. If FOR *filename* is omitted, the entire directory is listed.
listfile	is the name of a file or output device such as :SP:, :LP:, or :F2:LIST.FIL, where the directory listing will be displayed. If TO *listfile* is omitted, the listing is displayed on the screen.
switches	are one or more of the following, separated by spaces:
0-9	Lists the directory. If omitted, 0 is assumed. If more than one directory number is specified, only the rightmost one has effect. The directory number also overrides any device specification in FOR *filename*.
	Lists the extended directory filename, number of bytes in the file, the owner's name, file type (directory or data), owner access rights, and world access rights.
SP	Lists the contents of the network spooler print queue when the workstation is connected to the network.
O	Prints the directory in a single column format. The default is a triple column format.

Description

The DIR command lists the contents of a specific directory.

NOTE

The wild card search matches only valid ISIS filenames. A directory name that does not conform to these criteria will not be found in a wild card search.

```
DIR of pathname or directory identifier
FILENAME OWNER LENGTH TYPE OWNER ACCESS WORLD ACCESS
```

where

FILENAME	lists the name of all of the data and directory files that resides in this directory.
OWNER:	consists of the following:
username	is the username of the file owner.
SYSTEM	is operating system and command files.
SUPERUSER	is files that belong to the Superuser.
NOT FOUND	is files assigned to a user whose username has been deleted from the system.
LENGTH	is the number of bytes the file takes up on the mass storage device.
TYPE	identifies the file as a data or directory file.
OWNER ACCESS	lists the access rights of the file owner.
WORLD ACCESS	lists the access rights for other users set by the file owner.

Examples

1. This example lists the extended directory /PROJ.C to which the directory identifier :F1: has been assigned.

```
-DIR 1 E<cr>
 DIR of /PROJ.C
FILE NAME    OWNER        LENGTH  TYPE  OWNER ACCESS  WORLD ACCESS
MEMOS.DIR   SANDI          10904   DIR     D R W          R
MYPROG.SRC  RITA            7299   DATA    D R            R
WORKFL.NEW  JEANNE          8063   DATA      R W          R
SYSTEM.LIB  SUPERUSER       3128   DATA    D R W          R
A           USER1           7299   DIR     D R W        D R W
ACC.DIR     JEANNE         10904   DATA    D   W        W
PROG.X12    DENNIS         10054   DATA    D R W          R
FMHP.CSD    AMY              210   DATA    D R            R
ALTER.86    SUPERUSER      27467   DATA      R          D R W
```

2. This example displays a directory of the spooler printer queue when the workstation is connected to the network.

```
-DIR SP<cr>
REMOTE SPOOLER
FILE NAME    OWNER        LENGTH  TYPE  OWNER ACCESS  WORLD ACCESS
9E.DIR       A            00123   DATA    D R W
```

3. This example copies the spooler directory to a file :F1:PRINT.LST, and prints it at the spooler printer when the workstation is connected to the network.

```
-DIR SP TO :F1:PRINT.LST<cr>
-COPY :F1:PRINT.LST TO :SP:<cr>
```

REMOVE — Delete a Directory

Syntax

```
         { :Fn:      }
REMOVE   {           } directory name<cr>
         { pathname/ }
```

where

:F*n*:	is the directory identifier assigned to the parent directory of the file to be removed.
pathname	is the complete fully qualified pathname of the parent directory.
directory name	is the name of the directory to be removed.

Description

The REMOVE command removes an empty directory file from the parent directory. To remove a directory file from the file system, the directory file must be empty and you must have access to the file.

The sample file structure in figure 2-1 has an empty directory file DIRA. To remove this file from the file system, type

```
-REMOVE /WINCH1.VOL/ISIS.SYS/DIRA<cr>
/WINCH1.VOL/ISIS.SYS/DIRA, REMOVED
```

or

```
-ASSIGN :F3:TO /WINCH1.VOL/ISIS.SYS<cr>
-REMOVE :F3:DIRA<cr>
:F3:DIRA, REMOVED
```

Possible Error Conditions

An error will result when

- A directory file is not empty.
- Any user attempts to remove a directory file and does not have delete access rights to the file.

Examples

1. This example shows a successful attempt to remove an empty directory file from the file system.

```
-REMOVE /WINCH1.VOL/DIRB/DIRC<cr>
/WINCH1.VOL/DIRB/DIRC, REMOVED
```

2. This example shows an attempt to remove an empty directory without the proper access rights.

```
-REMOVE /WINCH1.VOL/PROJECT.DIR<cr>
/WINCH1.VOL/PROJECT.DIR,
iNDX FILE ACCESS RIGHTS VIOLATION
```

3. This example shows an attempt to remove a directory file that is not empty.

```
-REMOVE /WINCH1.VOL/ISIS.SYS<cr>
/WINCH1.VOL/ISIS.SYS,
ATTEMPT TO DELETE A NON-EMPTY DIRECTORY
```

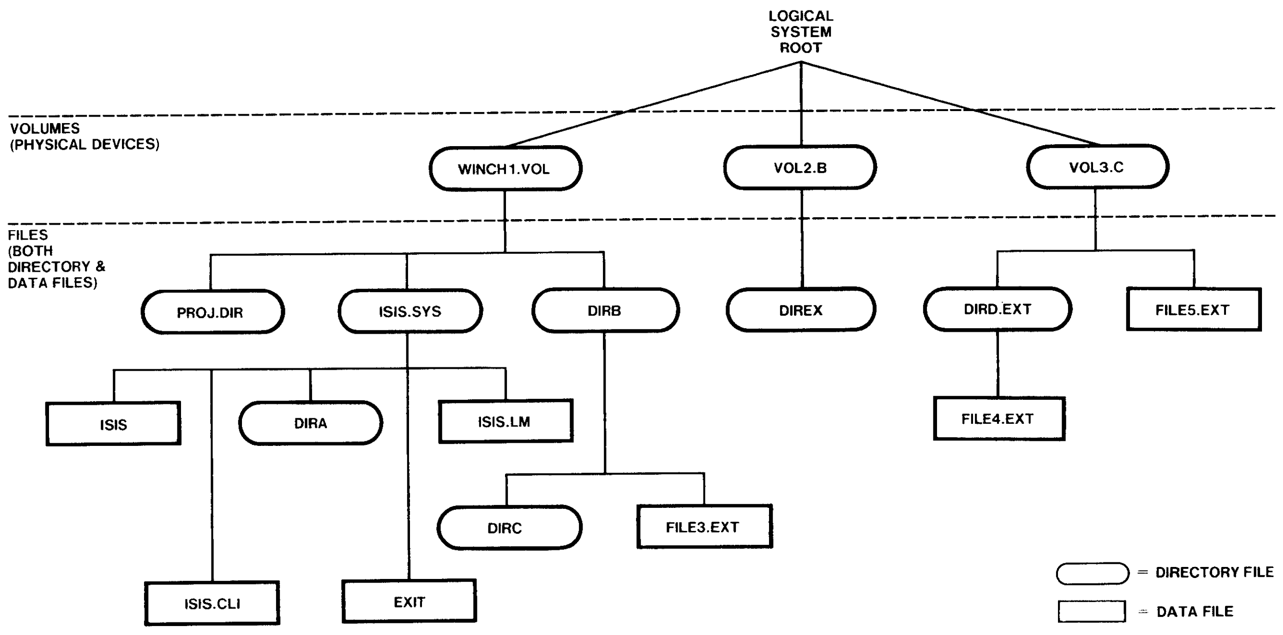


Figure 2-1. Hierarchical File Structure

121880-2

RENAME — Rename a File

Syntax

```
RENAME [:Fn:]oldname TO [:Fn:]newname<cr>
```

where

:F*n*:	refers to the directory identifier of the directory where the file resides, and must be the same for both *oldname* and *newname*.
oldname	is the name of an existing file to which you have been granted delete access rights. *oldname* follows :F*n*: with no intervening space, as in :F2:MYPROG.
newname	is the new name to be assigned to *oldname*. The *newname* follows :F*n*: with no intervening space, as in :F2:PROG1.

Description

The RENAME command changes the name of a file.

Enter the RENAME command to change the name of an existing file to a new name that does not already exist; the system changes the directory. Wildcards cannot be used.

However, if another file with the new name already exists, the system displays the following message:

```
newname, ALREADY EXISTS, DELETE?
```

To delete the existing file, enter a Y or y followed by a carriage return. RENAME will delete the existing file and change the name of *oldname* in the directory.

If you have not been granted delete access rights to the existing file, or if you enter any character other than Y or y, the existing file is not deleted and the file to be renamed is not renamed.

Possible Error Conditions

An error occurs when

- *oldname* is a nonexistent file.
- The device is not assigned.
- You do not have proper access rights.

Examples

1. To rename a file, assign a directory identifier to the directory first.

```
-ASSIGN :F3: TO /VOLUME/MYBOOK<cr>
-RENAME :F3:CHAP1B TO :F3:CHAP2<cr>
```

2. This example illustrates an attempt to rename a file to which the user has not been granted delete access rights.

```
-RENAME NEWPRG.TXT TO PROGA.TXT<cr>
NEWPROG.TXT, iNDX FILE ACCESS RIGHTS VIOLATION
-
```

3. In this example, the new name is the name of an existing file.

```
-RENAME TEXT.BAK TO TEXT.OLD<cr>
TEXT.OLD, ALREADY EXISTS, DELETE? Y<cr>
-
```

4. This example shows an attempt to rename a file and move to another directory.

```
-RENAME :F6:OLDFILE TO :F7:NEWFIL
:F7:NEWFIL, NOT ON SAME DISK
-
```

SPACE — Display the Volume Information of the Specified File

Syntax

```
SPACE /volume name<cr>
```

where

/volume name	is the volume root directory (e.g., /VOL1.A) for the given physical device.

Description

The SPACE command returns information about the amount of available space on any given disk at that specific point in time. Information is returned in the following format:

```
VOLUME GRANULARITY = number (granularity = number of bytes/sector)
FREE BLOCKS        = number
TOTAL BLOCKS       = number
FILES AVAILABLE    = number
TOTAL FILES        = number

MM/DD/YY hh:mm:ss
```

where

MM/DD/YY hh:mm:ss	is the month, day, year, hour, minutes and seconds.
number	is a decimal integer.

Examples

1.
```
-SPACE /VOL1.A<cr>
```

The information is provided for the volume whose root directory is /VOL1.A.

NOTE

In a multiprogramming environment, the amount of space is volatile. Therefore, the information returned by the SPACE command should be considered as approximate.

VERS — Display ISIS Utility Program Version Numbers

Syntax

```
VERS command<cr>
```

where

command is one of the ISIS command programs.

Description

The VERS command lists the version number of the ISIS-IV command programs.

The VERS command identifies the version number of the different versions of ISIS IV command programs. Each version works correctly only with the corresponding version of ISIS. Other Intel software such as editors and translators do not contain version numbers.

Examples

1. This example successfully lists the version number of a compatible ISIS-IV command program.

```
-VERS DIR<cr>
V1.0N
```

2. This example shows an attempt to list the version number of a user file.

```
-VERS MYFILE.EXT<cr>
file does not contain a program version number
```

3. This example shows an attempt to list the version number of a file not in the directory.

```
-VERS NONFLE<cr>
ERROR 13 USER PC 375B
```

4. This example lists the version number of the command VERS.

```
-VERS VERS<cr>
V1.0N
```

WHO — Display the Name of the User

Syntax

```
WHO<cr>
```

Description

The WHO command displays the name of the user who is currently logged on. For example,

```
-WHO<cr>
 I AM SUSAN
-
```

NOTE

If there is no user currently logged on, the system will display the following error message:

```
USER NOT LOGGED ON
-
```

Code Conversion Commands

The code conversion commands exist for two reasons:

- To provide compatibility with systems employing hexadecimal object file format
- To convert programs created under previous versions of ISIS

The programs called by the code conversion commands convert the character coding of these files but do not otherwise alter the information of the original programs.

HEXOBJ — Convert Hexadecimal Code to Absolute Object Code

Syntax

```
HEXOBJ hexfile TO absfile [START (addr)]<cr>
```

where

hexfile	is a file of machine object code in hexadecimal format.
absfile	is the output file from HEXOBJ containing the absolute object module that can be loaded for execution under ISIS-II, III(N), and IV. The module name stored with the object module is the same as the name part of the ***absfile*** filename. The absolute object module produced by HEXOBJ contains a symbol table if symbols were defined.
START *addr*	is used to include a starting address (the address of the first instruction to be executed) in the absolute module. The address can be specified by a hexadecimal, decimal, octal, or binary number followed by a letter indicating the base. The letter is H for hexadecimal, O or Q for octal, B for binary, D or omitted for decimal. Thus the address 4000 hexadecimal is specified as START (4000H). If START *addr* is omitted, the starting address is taken from the end-of-file record of the hexadecimal format file, which is determined by the END assembly language statement. If no starting address is specified in any of the above ways, zero is assumed. An attempt to execute such a program will cause entry into the interactive monitor.

Description

The HEXOBJ command converts object code in hexadecimal format to an absolute object module compatible with ISIS-II, III(N), and IV. The hexadecimal format is produced by cross-product translators that run on large machines and by assemblers of earlier versions of ISIS.

Examples

1. This example converts a hex file to absolute object format, recording a starting address in the object module.

```
-HEXOBJ PRIME.HEX TO PRIME.OBJ START(3200H)<cr>
-
```

OBJHEX — Convert ISIS-II, III(N), and IV Absolute Object Code to Hexadecimal Code

Syntax

OBJHEX *absfile* TO *hexfile*<cr>

where

absfile	is the file containing an ISIS-II, III(N), or IV absolute object module.
hexfile	is the file to contain the hexadecimal object code converted from the ISIS-II, III(N), or IV format. The starting address (address of the first instruction to be executed) is taken from *absfile*. The hexadecimal object code produced by OBJHEX does not contain a symbol table.

Description

The OBJHEX command converts an ISIS-II, III(N), or IV absolute object module to hexadecimal format.

CHAPTER 3
USE OF ISIS-IV BY OTHER PROGRAMS

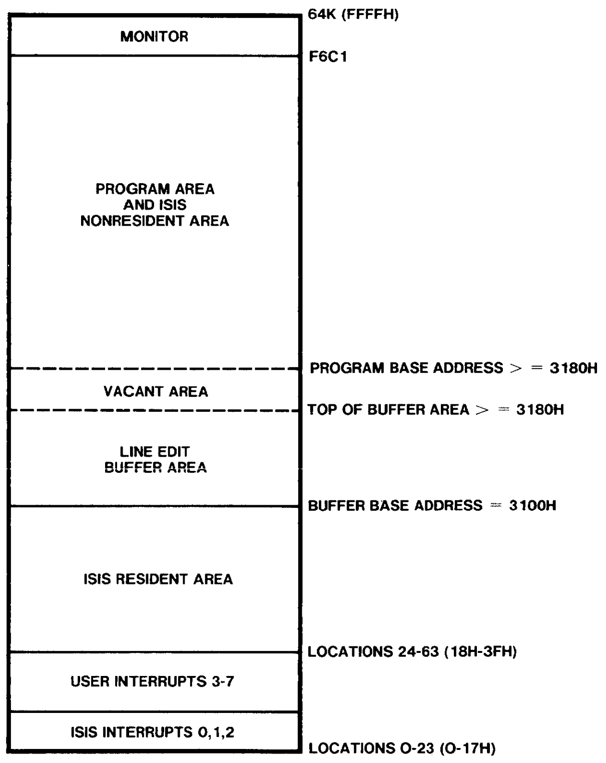
Writing programs that make use of the capabilities of ISIS-IV is the same as writing any other program except that the program includes ISIS-IV system calls. The placement of your program in memory must take into consideration how memory is organized under ISIS-IV.

Memory Organization And Allocation

The organization of Intellec memory under ISIS-IV is shown in figure 3-1.

Interrupt Vectors

Interrupts 0 through 2 are reserved for ISIS-IV. Interrupts 3 through 7 are available for use within the program. However, other Intel software products can also use interrupts 3 through 7. When using interrupts, you must be sure to avoid conflicts with Intel software. The locations 24 through 63 are the only locations below 3100H that can be loaded with user code. Loading other locations below 3100H is not allowed.



64K (FFFFH)
MONITOR
F6C1
PROGRAM AREA AND ISIS NONRESIDENT AREA
PROGRAM BASE ADDRESS > = 3180H
VACANT AREA
TOP OF BUFFER AREA > = 3180H
LINE EDIT BUFFER AREA
BUFFER BASE ADDRESS = 3100H
ISIS RESIDENT AREA
LOCATIONS 24-63 (18H-3FH)
USER INTERRUPTS 3-7
ISIS INTERRUPTS 0,1,2
LOCATIONS 0-23 (0-17H)

Figure 3-1. Intellec® Memory Organization 121980-3

The Kernel

The ISIS-IV resident area is reserved for the kernel, the part of ISIS-IV that is always resident in RAM memory. It may be viewed as a collection of subroutines. Although the kernel is protected from a program load operation, it is not protected from an executing program that may accidentally destroy the integrity of ISIS-IV by writing to this area. This will cause subsequent errors when system services are requested.

Input and Output to Files

User programs perform input/output (I/O) by making calls to the kernel, i.e., system calls. All I/O occurs to or from files and is status-driven rather than interrupt-driven. Interrupts 0, 1, and 2 are reserved for ISIS-IV and must not be masked or altered by your programs.

A file is an abstraction of an I/O device, considered to be a collection of information, usually in machine-readable form. A file can be formally defined as a sequence of eight-bit values called bytes.

ISIS-IV usually places no semantic interpretation on the byte values of a file, with the exception of lined files. Programmers, programs, and devices frequently interpret the bytes as representing ASCII values, and thereby characters.

Programs receive information by reading from an input file and transmit information by writing to an output file.

Every file is identified by a file name that has two parts: a filename and an optional extension, separated by a period (e.g., FILE3.EXT). A filename is a sequence of from one to six ASCII characters; an extension is a sequence of from one to three ASCII characters. To facilitate name specification within command strings, these ASCII characters are constrained to be letters and/or digits.

A major purpose of ISIS-IV is to enable ISIS users of Series IV to access local (private) and remote (public) iNDX files. For ISIS to access the iNDX files, the file must be identified by its fully qualified pathname. The pathname (e.g., /VOL1.A/DIRB/FILE3.EXT) identifies every volume and directory from the logical system root to the file. You must assign a directory identifier (e.g., :F1:) to the fully qualified pathname of the directory with the ASSIGN command before the file can be accessed.

```
-ASSIGN :F1: TO VOL1.A/DIRB<cr>
```

The pathname for the file FILE3.EXT is now :F1:FILE3.EXT.

For every non-disk device supported by ISIS-IV, there are one or more associated files, each identified by a name consisting of a pair of ASCII characters between colons (see Appendix D for a complete list).

No file can exist on more than one physical device. In particular, a disk file must reside entirely on one diskette/platter.

Three files (:BB:, :CI: and :CO:) deserve special mention:

ISIS-IV supports a virtual input/output device known as a Byte Bucket (:BB:). This device acts as an infinite sink for bytes when written to, and a file of zero length when read from. Multiple opening of :BB: is allowed; each open returns a different connection number (AFTN).

ISIS-IV supports a virtual console known as the Console that is implemented as two files, an input file (:CI:) and an output file (:CO:). These two files are always open. :CI: is always a lined file; :CO: is its associated echo file.

When you invoke ISIS, the assignments in table 3-1 for :CI: and :CO: are made by ISIS.

If :CO: is initialized to :BO: or :BB: and ISIS-IV is executing in the foreground, the following message appears:

```
WARNING: ISIS CONSOLE OUTPUT IS DIRECTED TO THE SERIES IV
LOG FILE
```

Whenever an end of file is encountered on :CI:, a fatal error will occur.

The Command Line Interpreter (CLI) always obtains its command lines from the current console. Briefly, each command contains a generalized keyword that specifies either a user program or CLI command. If the former, CLI causes that program to be loaded and run; otherwise CLI performs the indicated command and reads another command line.

Table 3-1. Console Assignment by ISIS

Invocation from	**Initial assignment of :CI: made to**
Keyboard	:VI: (video terminal keyboard)
Command file	:BI: (batch input)
Invocation from	**Initial assignment of :CO: made to**
Foreground (but not import) without a Series IV LOG file	:VO: (video terminal screen)
Foreground (but not import) with a Series IV LOG file	:BO: (batch output)
Background or import with a Series IV LOG file	:BB: (byte bucket)
Background or import with a Series IV LOG file	:BO:

Buffers

The buffer area is used by ISIS-IV for input/output buffers of 128 bytes each. One permanent buffer is used by ISIS-IV for console input/output. Other buffers are allocated and deallocated dynamically by ISIS-IV according to the input/ output requirements of the program. These requirements come from your explicit system calls to ISIS-IV or are derived from your source code by the translator (e.g., PL/M or assembler).

The minimum size of the buffer area is 128 bytes, allowing for one buffer, the ISIS-IV permanent buffer. If the program requires more than one buffer, the buffer area increases at the expense of the vacant area.

Computing Program Base Address

Program Area

The program area is above the buffer area. It is used alternately by the programs and ISIS-IV.

Nonresident ISIS-IV routines (all commands except DEBUG) run in the program area. These include the command interpreter, the editor, assembler, compiler, linker, locater, and library manager. Whenever you communicate with ISIS-IV via console commands, you are using the nonresident command interpreter running in the program area. These nonresident ISIS-IV routines may use all available RAM for buffers.

Monitor Area

The Monitor occupies memory from F6C1H to FFFFH for its workspace.

The Monitor MEMCK routine can be called if a program needs to know the highest available location of contiguous RAM (below the Monitor workspace).

Base Address of Your Program

You determine the base address of the program by commands to LOCATE (or by an ORG statement in 8080/8085 assembly language absolute programs).

Before deciding the base address, you must determine the maximum area required for buffers under ISIS-IV. The number of buffers varies during execution, but the buffer area must be large enough for the maximum number of buffers allocated simultaneously.

If you locate the base address of your program below 3180H (or allocate less than one buffer), an error message is generated.

The program base address can be calculated using the following formula:

12,288 + (128 * *N*)

where

N	is the maximum number of buffers required simultaneously by the program. Use the following rule to determine *N*: An open line-edited file (e.g., :CI:) requires one buffer until the file is closed.

General Parameter Discussion

The parameters to system service routines presume certain uses. If your understand the intended usage, you will readily see the reasons for the parameters and how to specify them to achieve your purposes.

The easiest one to understand, used by every routine, is ***status$p***. This address is filled with a non-zero error code if the service operation could not complete its task normally. (Appendix A gives these codes and their meanings.)

Arguments

To invoke the execution of a program, you can type its name at the console, e.g., PROGRM. PROGRM may have options that can be specified on the invocation line. If so, the remainder of that line (after the program name) is called a command tail.

This command tail is accessible to PROGRM via ISIS-IV system calls to READ or RESCAN. These routines can handle, in similar fashion, any file that you create to be read as if it were the console. These are called lined or line-edited files and are discussed later. The PROGRM using them can then alter its mode of operation depending on the options specified with each invocation, adding flexibility to the user-interface of the programs.

Connections

ISIS-IV maintains a list of six devices or files that the program may use during its execution, i.e., a list of connections. A connection is a word, named and declared by you, filled by the system service routine OPEN.

You then use this word to specify that file or device whenever you need to perform any operation on it, i.e., to read, write, seek, rescan, or close it.

The list is also called an Active File Table, and the entries on it are Active File Table Numbers, or AFTNs. Only objects on this list can be used for input/output operations. Such operations are accomplished using the connection rather than the actual device or file name. During execution, your program may perform these functions on multiple files. When I/O actions for a given file are complete or the file will not be needed for the next phase of program activity, it can be closed to make room for other files (no more than six) that may be needed sooner.

Input/Output Parameters

In order to perform a READ or a WRITE, several questions must be answered:

1. How many bytes are to be transferred?
2. To (or from) what file?
3. From (or to) what memory locations?

In the descriptions that follow, (1) is usually supplied as the parameter *count*, and (3) as the parameter *buffer$p*, the address of the locations to be read from or written into.

Question (2) naturally requires an ISIS-IV pathname, e.g., :F1:YOURDA.TA1, that can be up to 14 characters long in the format shown. This format for a string differs from the one used under the iNDX operating system in two respects: it does not begin with a count of the characters to follow, and it must end with a nonvalid pathname character. ISIS-IV interprets the first nonvalid pathname character encountered as marking the end of the string. Thus the last character before the nonvalid character is the last character in the string.

Rather than require you to give the name of each file every time you do any input/output, ISIS-IV maintains the table of active files mentioned previously. One call to OPEN establishes the full name as an entry in this table and returns to you a connection number for your use in all further references to this file while it is in use.

Terms

If you have used earlier ISIS systems, you will notice a change in the terms describing some parameters for system calls. The parameters have not changed. The new terms highlight similarities between ISIS-II, ISIS-III(N), and ISIS-IV in both concept and usage.

In several cases the change is simply appending the characters *$p* to each term that is actually used as an address rather than as the value stored at that address. In the PL/M calls to the routines, you simply use the dot operator to provide the address of the variable you declared for use in these routines.

In other cases, there is a new name. For example, *conn* is used instead of AFTN to represent a connection to a file (formerly called an active file table number, as described previously). The pointer to this connection, giving its address, is called *conn$p* where it was formerly named AFTNPTR.

The syntax charts introducing each functional grouping of commands show the placement of each parameter, and the examples given with each individual command illustrate actual declaration and usage.

Line-Edited Input Files

ISIS-IV provides a special way of reading ASCII files called line-editing. Line-editing was designed for (but not restricted to) the case of a human user, prone to err, typing characters at a keyboard. The rubout key and control characters allow you to correct istakes and then transmit a perfect line by typing a carriage return to which a line feed is appended automatically.

You tell ISIS-IV that a file is to be line-edited by supplying a parameter in the OPEN system call. This must also specify an echo file you opened earlier because every line-edited file must have an file to which ISIS-IV sends an echo of the input. If no echo is desired, the byte bucket file :BB: can be specified as the echo file.

Terminating a Line

While a line is being physically entered from an input device, it is accumulated by ISIS-IV in a 122-character line-editing buffer. The contents of the buffer are altered by ISIS-IV when an editing character is entered. An explanation of editing characters follows. No data is transferred to the requesting program until the line is terminated in one of four ways:

- A carriage return is entered (automatically appended to every line feed).
- A line feed is entered.
- An escape is entered.
- A non-editing character is entered as the 122nd character.

Reading from the Line-Edit Buffer

When the line has been terminated, the next (or pending) READ system call transfers data from the line-editing buffer to the buffer of the requesting program. ISIS-IV maintains a pointer to keep track of what characters have been read from the line-editing buffer. For example, if the line-editing buffer contains 100 characters and you issue a READ system call with a COUNT 50, the first 50 characters are transferred to the buffer of the program and the buffer pointer is moved to the fifty-first character. The next READ system call transfers characters starting at the fifty-first character.

If the READ system call requests 100 bytes and the line-editing buffer contains 50, only 50 bytes are transferred.

A READ call returns bytes from only one logical line at a time. This means only up to 122 "uncancelled" characters. Thus, READ's of line-edited files often transfer fewer bytes than requested by *count*.

A READ system call returns no characters from a logical line until the line has been input in its entirety. Thus, during physical input, the logical line is accumulated in an internal buffer; no information in the buffer is transferred to the reading program until the termination character (normally a < cr > is seen. Therefore, ISIS-IV has the opportunity to modify buffer contents conditionally on values entering the buffer.

When all the characters in the line-editing buffer have been read, the buffer pointer is positioned after the last character; the buffer is not cleared yet. In fact, the RESCAN system call can be used to reposition the buffer pointer to the beginning of the line-editing buffer so subsequent READ system calls can reread the contents.

When the buffer has been completely read (the pointer is after the last character), a new READ system call will transfer new input from the line-edited file into the line-editing buffer. When the line is terminated, the number of characters requested by READ is transferred to the program.

Editing Characters

The characters in table 3-2 are used to edit the input of a line-edited file. Control characters are entered by holding down the control key (CTRL) while the character is typed.

Table 3-2. Editing Keys

Key	Action
RUBOUT	Deletes preceding character from buffer.
CONTROL-P	Causes next character typed to be entered literally in line-editing buffer. Use when you want editing character or terminating character to be entered in buffer rather than to cause its usual editing or terminating function.
CONTROL-R	Causes carriage return/line feed to be sent to console output device, followed by current undeleted contents of buffer. No other effects.
CONTROL-X	Causes entire contents of buffer to be deleted, including itself. Echoed as #, carriage return, line feed.
CONTROL-Z	Is only way to indicate end-of-file from keyboard input device. Acts like control-X except has no echo and causes READ system call to return immediately without transferring any characters, thus simulating end-of-file. If more characters are entered after control-Z, they are entered in line-editing buffer and can be read by subsequent READ system call.

Reading a Command Line

Reading a command line from the console input device is a special case of reading a line-edited file.

When a command is entered at the console, it is collected by ISIS-IV in the line-editing buffer for :CI: and is not available to the command interpreter (a nonresident ISIS-IV routine) until it is terminated. The command interpreter reads only the command name and then calls the program with that name, leaving the line-editing buffer pointer positioned after the command name. The loaded program can issue a READ, which transfers data starting with the first parameter, or the program can issue a RESCAN to position the pointer to the beginning of the buffer so it can also read the command name.

For example, suppose the following command has been entered:

```
-TYPE :F1:PROGA.SRC<cr>
```

The line-editing buffer for :CI: contains 20 characters as follows:

T	Y	P	E		:	F	1	:	P	R	O	G	A	.	S	C	R	CR	LF
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

When the TYPE program is loaded, the buffer pointer is at the fifth character (the space following TYPE). A READ call starts transferring characters at the fifth character. New input from :CI: to the line-edit buffer does not happen until the buffer pointer is moved to the end of the buffer (after the 20th character) and a READ call is issued.

Remember that when control is passed to the loaded program, the buffer pointer is positioned after the command name, not at the end of the buffer. In the preceding example the buffer pointer is positioned at the space after TYPE. If no parameters are passed, the first READ from :CI: returns the carriage return/line feed left from the command line. For example, suppose the following command has been entered:

```
-PROGA.BIN<cr>
```

and the line-editing buffer contains 11 characters as follows:

P	R	O	G	A	.	B	I	N	CR	LF
1	2	3	4	5	6	7	8	9	10	11

When PROGA.BIN is loaded, the pointer is at the carriage return. If subsequent input is expected from the console input device, an extra READ must first be issued to clear the buffer of the carriage return/line feed.

If the program does not read from :CI:, the remaining carriage return/line feed is cleared by ISIS-IV from the buffer before a new command is read by the command interpreter (e.g., after PROGA.BIN exits).

Summary of System Calls

The ISIS-IV services that can be called by the program include the following:

- Input/output operations for the mass storage devices, keyboard, CRT, and the standard Intellec peripherals, except the Universal PROM Programmer
- Directory maintenance (ATTRIB, DELETE, GETATT, GETD, RENAME)
- Console device assignment and error message output (CONSOL, ERROR, WHOCON)
- Program loading and execution and return to the supervisor (EXIT, LOAD)

The interface to these services is a call to ISIS-IV that specifies the services desired and the address of the parameter list the supervisor is to access. The specific calling sequences are described with the call descriptions. Note that an ISIS-IV call uses your stack. Your stack must have the depth necessary to handle the call. A call to ISIS-IV destroys the contents of the CPU registers.

The system calls are described in terms of their operation and the parameters the program must supply.

To clarify the effect of certain system calls on your files, two integer quantities, LENGTH and MARKER, are associated with each file in this description. LENGTH is the number of bytes in the file. MARKER is the number of bytes already read or written in the file (that is, it acts as a file pointer).

System Call Syntax and Usage

The ISIS-IV system calls can be called from a PL/M or Assembler Language program. If the program makes an ISIS-IV system call, link the object program with SYSTEM.LIB using the LINK program.

SYSTEM.LIB contains the procedures necessary to interface your programs containing ISIS-IV system calls with the ISIS-IV system.

PL/M Calls

Any PL/M program can interface to ISIS by performing calls to procedures in SYSTEM.LIB. The program must include external procedure declarations so the proper procedures from SYSTEM.LIB will be included with the program by LINK. These external procedure declarations may be declared as type ADDRESS, but may also be values as well as addresses of values.

Assembler Language Calls

The interface between the 8080/8085 Assembler Language program and ISIS is accomplished by calling a single ISIS entry point (labeled ISIS) and passing two parameters. The first parameter is a number that identifies the system call; the second is the address of a control block that contains the additional parameters required by the system call. The first parameter is passed in register C and the address of the control block is passed in the register pair DE. The entry point must be defined in the program as an external:

```
EXTRN ISIS
```

The ISIS entry point is defined in a routine in SYSTEM.LIB that must be included in the program. When using LINK, specify the name of the program followed by the name SYSTEM.LIB. See the *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems* (121617) for more information on LINK.

Syntax charts that introduce each functional grouping of system calls show the placement of each parameter, and the examples given with each individual system call illustrate actual declaration and usage.

System Calls Cautions

CAUTION

ISIS-IV is non-reentrant. Consequently, an interrupt service routine that is activated while ISIS-IV is executing must not call ISIS-IV.

ISIS-IV references files by a connection number (AFTN, or active file table number). Be careful not to confuse the AFTN with the PL/M construction .AFTN. The period (.) specifies the address of the memory location where the AFTN is stored.

To reduce this potential confusion, both the syntax charts and the sample PL/M declarations in the examples refer to the connection number as *conn* and to the address as *conn$p*. Then in the CALLs of the PL/M examples, periods precede the names when addresses are to be supplied.

Similarly, the charts and declarations show *path$p* to represent the address of a memory location containing the string naming a device or file. The CALL then shows a period before the variable you declared to hold that string.

File Input/Output Calls

Seven system calls are available to your program for controlling file input/output (see figure 3-2). These subroutines let you open files for read or write operations, move the pointer in an open file, and close the files when you are finished.

These data transferring services of the supervisor transfer variable-length blocks of data between standard peripheral devices and a memory buffer area in the program. These calls establish and maintain the MARKER and LENGTH quantities associated with the file being operated upon.

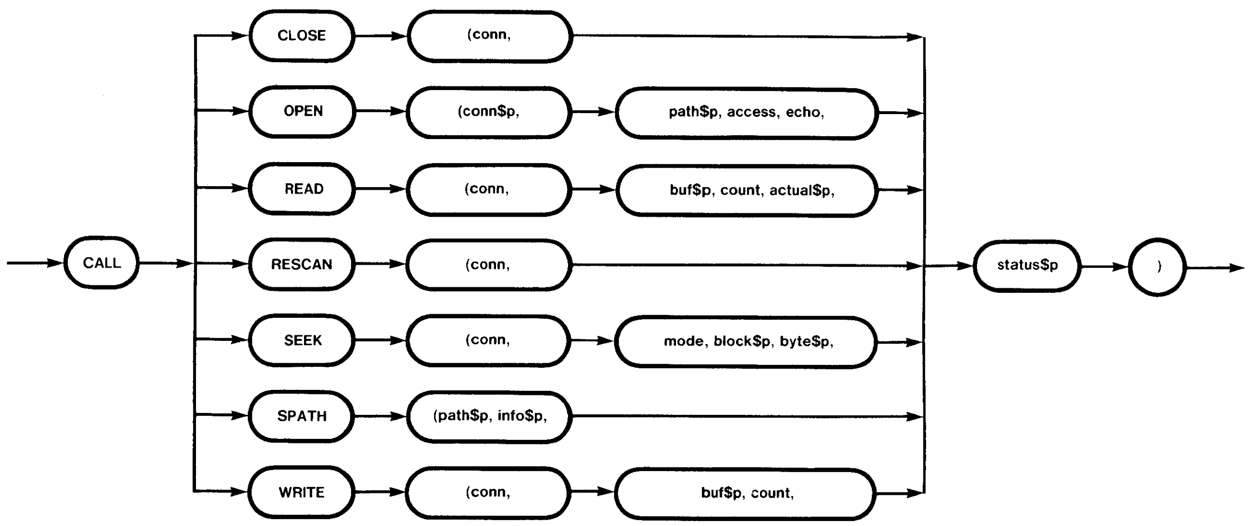


Figure 3-2. File Input-Output System Calls 121980-4

CLOSE — Terminate Input/Output Operations on a File

Syntax

`CALL CLOSE` *(conn, status$p)*

You must`pass two parameters, *conn* and *status$p*, in the CLOSE call,

where

conn	is the connection number (AFTN) of the file to be closed. The AFTN was returned by a preceding OPEN call.
status$p	is the address of a memory lcation for the return of non-fatal error numbers. The non-fatal error numbers issued by CLOSE are listed in Appendix A.

The CLOSE call removes a file from the system input/output tables and releases the buffers allocated for it by OPEN. All files should be closed when input/output processing is complete.

Examples

1. PL/M CLOSE CALL

```
CLOSE:
 PROCEDURE (conn, status$p) EXTERNAL;
    DECLARE (conn, status$p) ADDRESS;
 END CLOSE;
.
.
.
DECLARE AFT$IN ADDRESS;
DECLARE STATUS ADDRESS;
.
.
.
CALL CLOSE (AFT$IN,.STATUS);
IF STATUS <> 0 THEN...
.
.
.
```

2. Assembly Language CLOSE Call

```
        EXTRN   ISIS            ; LINK TO ISIS ENTRY POINT
CLOSE   EQU     1               ; SYSTEM CALL IDENTIFIER

        MVI     C,CLOSE         ; LOAD IDENTIFIER
        LXI     D,CBLK          ; ADDRESS OF PARAMETER
                                ; BLOCK
        CALL    ISIS
        LDA     CSTAT           ; TEST ERROR STATUS
        ORA     A
        JNZ     EXCEPT          ; BRANCH TO EXCEPTION
                                ; ROUTINE
;
CBLK:                           ; PARAMETER BLOCK FOR
                                ; CLOSE
CAFT:   DS      2               ; FILE AFTN
        DW      CSTAT           ; POINTER TO STATUS
;
CSTAT:  DS      2               ; STATUS (RETURNED)
;
```

OPEN — Initialize File for Input/Output Operations

Syntax

`CALL OPEN` *(conn$p, path$p, access, echo, status$p)*

You must pass five parameters, *conn$p, path$p, access, echo*, and *status$p*, in the OPEN call,

where

conn$p	is an address of a two byte field in which ISIS will store the active file number (AFTN) of the file that is opened. Your program will use this value for other calls relative to this file. :CI: and :CO: are always open and have the AFTNs 1 and 0 permanently assigned. Excluding :CI: and :CO:, you can only have six files open at any one time. Be careful not to confuse AFTN with the PL/M construction .AFTN. The period prefacing .AFTN signifies AFTN's location in memory.
path$p	is the address of the ASCII string containing the name of the file to be opened. The ASCII string can contain leading space characters but no embedded space characters. It must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used.
access	is a value indicating the access mode for which the file is being opened. A value of 1 specifies that the file is open only for input to the system READ. MARKER is set to 0 and LENGTH is unchanged. If the file is nonexistent, a non-fatal error occurs. A value of 2 specifies that the file is open only for output from the system: WRITE.MARKER and LENGTH are set to 0. If the file is nonexistent, a file with the default iNDX access rights (e.g., WORLD READ, WRITE, and DELETE on a 5¼-inch miniature flexible diskette, and OWNER READ, WRITE, DELETE for all other devices) is created. The filename is specified at location *path$p*. If it already exists, information in the file will be overwritten. A value of 3 specifies that the file is open for update: READ and WRITE. MARKER is set to 0. LENGTH is unchanged for existing files and set to 0 for new files. If the file is nonexistent, a new file with the default iNDX access rights is created. The filename is specified at location *path$p*.
echo	is the AFTN of the echo file if the file is to be opened for line editing. The echo file must be previously opened for output (access=2). The AFTN of the echo file is passed in the least significant byte of the field. If this field contains 0, no line editing is done. To specify an AFTN of 0 for :CO:, a nonzero value must be in the most significant byte and zero in the least significant byte. For example, FF00H specifies the AFTN for the :CO: device.
status$p	is the address of a memory location for the return of non-fatal error numbers. The error numbers that OPEN can return are listed in Appendix A.

Description

The OPEN call initializes ISIS tables and allocates buffers that are required for input/output processing of the specified file.

Possible Error Conditions

If you open a file for an access mode that is not physically possible, a non-fatal error will occur. The following will cause non-fatal errors:

- Opening :VI: (video terminal keyboard) or :VO: (video terminal screen) when you are executing ISIS-IV under Background or Import
- Opening :BI: (batch input) when you have interactively invoked ISIS-IV. You must be executing ISIS-IV under Background, SUBMIT, or Import to open :BI:.
- Opening :BO: (batch output) when you have no Series IV LOG file at the time that you invoked ISIS-IV
- Opening a file that begins with an unassigned directory identifier (e.g., :F*n*: has not been identified)
- Open a file that begins with :SP: (spool printer) when you have not defined :SP: as a logical name, or if the SPOOL directory is not present within the volume root directory of the system device

If you do not have appropriate access rights to the specified file (e.g., opening for read without read access rights), a non-fatal error will result.

Examples

1. PL/M OPEN Call

```
OPEN:
 PROCEDURE (conn$p, path$p, access, echo, status$p) EXTERNAL;
  DECLARE (conn$p, path$p, access, echo, status$p) ADDRESS;
 END OPEN;
 .
 .
 .
 DECLARE AFT$IN ADDRESS;
 DECLARE FILENAME(15) BYTE DATA (':F1:MYPROG.SRC');
 DECLARE STATUS ADDRESS;
 .
 .
 .
CALL OPEN(.AFT$IN,.FILENAME,1,0,.STATUS);
IF STATUS<>0 THEN...
```

2. Assembly Language OPEN Call

```
        EXTRN   ISIS            ; LINK TO ISIS ENTRY POINT
OPEN    EQU     0               ; SYSTEM CALL IDENTIFIER

        MVI     C,OPEN          ; LOAD IDENTIFIER
        LXI     D,OBLK          ; ADDRESS OF PARAMETERS
                                ; BLOCK
        CALL    ISIS
        LDA     OSTAT           ; TEST ERROR STATUS
        ORA     A
        JNZ     EXCEPT          ; BRANCH TO EXCEPTION
                                ; ROUTINE
.
.
.
OBLK:                           ; PARAMETER BLOCK FOR OPEN
        DW      OAFT            ; POINTER TO AFTN
        DW      OFILE           ; POINTER TO FILENAME
ACCESS: DW      1               ; ACCESS, READ=1, WRITE=2,
                                ; UPDATE=3
ECHO:   DW      0               ; IF ECHO <> 0,
                                ; ECHO = AFTN OF
                                ; ECHO OUTPUT FILE
        DW      OSTAT           ; POINTER TO STATUS
;
OAFT:   DS      2               ; AFTN (RETURNED)
OSTAT:  DS      2               ; STATUS (RETURNED)
OFILE:  DB      ':F0:FILE.EXT ' ; FILE TO BE OPENED
```

READ — Transfer Data from File to Memory

Syntax

`CALL READ` *(conn, buf$p, count, actual$p, status$p)*

You must pass five parameters, *conn, buf$p, count, actual$p,* and *status$p*, in the READ call,

where

conn	is the connection number (AFTN) of a file that is open for input or update. The AFTN is returned by a preceding OPEN call or is 1 for :CI:
buf$p	is the address of a buffer to contain the data read from the open file. The buffer must be at least as long as the count described below. If the buffer is too short, the memory locations that follow the buffer will be overwritten.
count	is the number of bytes to be transferred from the file to the buffer.
actual$p	is the address of a memory location in which ISIS will store the actual number of bytes successfully transferred. The same number is added to MARKER. The actual number of bytes transferred is never more than the number specified in the count parameter above. For line-edited files the actual number of bytes is never more than the number of bytes in the line-edit buffer. When a file is not line edited, the number of bytes is equal either to count or to LENGTH minus MARKER, whichever is fewer. *actual* = 0 is the only reliable way of detecting end-of-file (for both lined and nonlined files).
status$p	is the address of a memory location in which ISIS will store non-fatal error numbers. The error numbers returned by the READ call are listed in Appendix A.

Description

The READ call transfers data from an open file to a memory location specified by the calling program. See "Line-Edited Input Files" in this chapter for information concerning reading line-edited files.

Examples

1. PL/M READ Call

```
 READ:
  PROCEDURE(conn, buf$p, count, actual$p, status$p)EXTERNAL;
    DECLARE (conn, buf$p, count, actual$p, status$p) ADDRESS;
  END READ;
.
.
.
DECLARE AFT$IN ADDRESS;
DECLARE BUFFER(128) BYTE;
DECLARE ACTUAL ADDRESS;
DECLARE STATUS ADDRESS;
.
.
.
CALL READ (AFT$IN,.BUFFER,128,.ACTUAL,.STATUS);
IF STATUS <> 0 THEN ...
```

2. Assembly Language READ Call

```
        EXTRN   ISIS          ; LINK TO ISIS ENTRY POINT
READ    EQU     3             ; SYSTEM CALL IDENTIFIER

        MVI     C,READ        ; LOAD IDENTIFIER
        LXI     D,RBLK        ; ADDRESS OF PARAMETER
                              ; BLOCK
        CALL    ISIS
        LDA     RSTAT         ; TEST ERROR STATUS
        ORA     A
        JNZ     EXCEPT        ; BRANCH TO EXCEPTION
                              ; ROUTINE
.
.
.
RBLK:                         ; PARAMETER BLOCK FOR READ
RAFT:   DS      2             ; FILE AFTN
        DW      IBUF          ; ADDRESS OF INPUT BUFFER
RCNT:   DW      128           ; LENGTH OF READ REQUESTED
        DW      ACTUAL        ; POINTER TO ACTUAL
        DW      RSTAT         ; POINTER TO STATUS
;
ACTUAL: DS      2             ; COUNT OF BYTES READ
                              ; (RETURNED)
RSTAT:  DS      2             ; STATUS (RETURNED)
IBUF:   DS      128           ; INPUT BUFFER;
```

RESCAN — Position MARKER to Beginning of Line

Syntax

`CALL RESCAN` *(conn, status$p)*

You must pass two parameters, *conn* and *status$p*, in the RESCAN call,

where

conn	is the connection number (AFTN) of a file open for line-edited input (echo file AFTN specified) by a preceding OPEN call.
status$p	is the address of a memory location for the return of non-fatal error numbers. The error numbers returned by the RESCAN call are listed in Appendix A.

Description

The RESCAN call is used on line-edited files only. It allows your program to move the MARKER to the beginning of a logical line that has already been read. Thus the next READ call starts at the beginning of the last logical line read. This line is not echoed (output to the echo file); it has already been input from the keyboard and echoed. Thus, the subsequent READ does not input from a file but only reads from a buffer in memory.

Examples

1. **PL/M RESCAN Call**

```
RESCAN:
 PROCEDURE (conn, status$p) EXTERNAL;
    DECLARE (conn, status$p) ADDRESS;
 END RESCAN;
.
.
.
DECLARE AFT$IN ADDRESS;
DECLARE STATUS ADDRESS;
.
.
.
CALL RESCAN (AFT$IN,.STATUS);
IF STATUS <> 0 THEN ...
.
.
.
```

2. Assembly Language RESCAN Call

```
        EXTRN   ISIS            ; LINK TO ISIS ENTRY POINT
RESCAN  EQU     11              ; SYSTEM CALL IDENTIFIER

        MVI     C,RESCAN        ; LOAD IDENTIFIER
        LXI     D,IBLK          ; ADDRESS OF PARAMETER
                                ; BLOCK
        CALL    ISIS
        LDA     ISTAT           ; TEST ERROR STATUS
        ORA     A
        JNZ     EXCEPT          ; BRANCH TO EXCEPTION
                                ; ROUTINE
.
.
.
IBLK:                           ; PARAMETER BLOCK FOR
                                ; RESCAN
IAFT:   DS      2               ; AFTN FROM OPEN
        DW      ISTAT           ; POINTER TO STATUS
;
ISTAT:  DS      2               ; STATUS (RETURNED)
;
```

SEEK — Position Disk File Marker

Syntax

```
CALL SEEK (conn, mode, block$p, byte$p, status$p)
```

You must pass five parameters, *conn, mode, block$p, byte$p*, and *status$p*, in the SEEK call,

where

conn	is the connection number (AFTN) of a file on a random access device opened for update or input. This connection was returned by a preceding OPEN call.
mode	is a value from 0 through 4 that indicates what action should be performed on the MARKER. The block and byte parameters (see below) are used either to represent the current MARKER position or to calculate the desired offset, depending on *mode*. A detailed discussion follows this parameter list.
block$p	is the address of a memory location containing a 2-byte value used for the block number. A block is equal to 128 bytes.
byte$p	is the address of a memory location containing a 2-byte value used for the byte number. The byte number may be greater than 128.
status$p	is the address of a memory location you declared to receive non-fatal error numbers from ISIS. The error numbers returned by the SEEK call are listed in Appendix A.

Description

The SEEK call allows your program to find the location of or to change the value of MARKER associated with a file open for read or update. The SEEK call can be used only with a file open for update or read. The MARKER can be changed in four ways: moved forward, moved backward, moved to a specific location, or moved to the end of the file. A nonfatal error occurs if SEEK is issued for a file opened for output.

Return Marker Location: Mode=0. Under this mode the system returns a pair of block and byte values (at *block$p* and *byte$p*) that signify the current position of the marker. For example, if the marker is just beyond the first block of the file, the system might return the numbers 1 and 0 in the addresses assigned to block and byte, respectively. It might also return the numbers 0 and 128, which point to the same byte in the file. The value of MARKER is given by the following equation:

```
MARKER = 128* (block number) + byte number
```

Move Marker Backward: Mode=1. If the mode value is 1, the marker is moved backward, toward the beginning of the file. The block and byte parameters determine the offset; for example, if block is equal to 0 and byte is equal to 382, the marker is moved backward 382 bytes. To define an offset of *N*, use block and byte values such that

```
N = 128* (block number) + byte number
```

If *N* is greater than MARKER, i.e., if the prescribed action would place the marker before the beginning of the file, MARKER is set to 0 (beginning of file), and a non-fatal error occurs.

Move Marker to Specific Location: Mode=2. If the mode value is 2, the marker is moved to a specific position in the file. The block and byte parameters define the position; for example, if block is equal to 27 and byte is equal to 63, the marker will be moved to block 27, byte 63. Similarly, if both block and byte are equal to 0, the marker is moved to the beginning of the file. If the file is open for update and the prescribed action would place the marker beyond the end of the file, ASCII nulls (00HH) are added to the file to extend the file to the marker. (Thus, LENGTH becomes equal to MARKER.)

Move Marker Forward: Mode=3. If the mode value is 3, the marker is moved forward, toward the end of the file. The block and byte parameters define the offset *N*, according to the following equation:

N + 128* (*block number*) + *byte number*

If the file is open for update and the specified action would place the marker beyond the end of the file, ASCII nulls (00HH) are added to the file to extend the file to the marker. (Thus, LENGTH becomes equal to MARKER.)

If the extension of a file by the SEEK operation causes an overflow on the mass storage device, a fatal error is reported, either during the execution of the SEEK call or when a program tries to write into the extended area of the file. This error can become evident at any time in the life of the file.

If an attempt is made to extend a file that is open only for input, the marker is set to the former end-of-file, and a non-fatal error occurs.

Move Marker to End of File: Mode=4. If the mode value is 4, the marker is moved to the end of the file. Block and byte parameters are ignored.

Examples

1. **PL/M SEEK Call**

```
SEEK:
 PROCEDURE (conn, mode, block$p, byte$p, status$p) EXTERNAL;
    DECLARE (conn, block$p, byte$p, status$p) ADDRESS;
 END SEEK;
.
.
.
DECLARE AFT$IN ADDRESS;
DECLARE BLOCKNO ADDRESS;
DECLARE BYTENO ADDRESS;
DECLARE STATUS ADDRESS;
.
.
.
CALL SEEK (AFT$IN,0,.BLOCKNO,.BYTENO,.STATUS);
IF STATUS<>0 THEN ...
.
.
.
```

2. **Assembly Language SEEK Call**

```
         EXTRN    ISIS           ; LINK TO ISIS ENTRY POINT
SEEK     EQU      5              ; SYSTEM CALL IDENTIFIER

         MVI      C,SEEK         ; LOAD IDENTIFIER
         LXI      D,SBLK         ; ADDRESS OF PARAMETER
                                 ; BLOCK
         CALL     ISIS
         LDA      SSTAT          ; TEST ERROR STATUS
         ORA      A
         JNZ      EXCEPT         ; BRANCH TO EXCEPTION
                                 ; ROUTINE
.
.
.
SBLK:                            ; PARAMETER BLOCK FOR SEEK
SAFT:    DS       2              ; AFTN FROM OPEN
MODE:    DS       2              ; TYPE OF SEEK
         DW       BLKS           ; POINTER TO BLKS
         DW       NBYTE          ; POINTER TO NBYTE
         DW       SSTAT          ; POINTER TO STATUS
;
BLKS:    DS       2              ; NUMBER OF SECTORS TO SKIP
NBYTE:   DS       2              ; NUMBER OF BYTES TO SKIP
SSTAT:   DS       2              ; STATUS (RETURNED)
;
```

SPATH — Obtain File Information

Syntax

`CALL SPATH` *(path$p, info$p, status$p)*

You must pass three parameters, *path$p, info$p,* and *status$p,* in the SPATH call,

where

path$p	is the address of an ASCII string containing the name of the file for which information is requested. The string can contain leading spaces but no embedded spaces. The string must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used.
info$p	is the address of a 12-byte memory location in which the system will return the information. After the call is completed, the buffer will contain the following information: Byte 0 - device number Bytes 1 through 6 - file name Bytes 7 through 9 - file name extension Byte 10 - device type Byte 11 - drive type
status$p	is the address of a memory location for a return of a non-fatal error number. The non-fatal error numbers issued by the SPATH call are listed in Appendix A.

The possible values for the contents of the *info$p* device number are

0 - directory 0
1 - directory 1
2 - directory 2
3 - directory 3
4 - directory 4
5 - directory 5
6 - teletype input (valid for ISIS-II and ISIS-III (N) only)
7 - teletype output (valid for ISIS-II and ISIS-III (N) only)
8 - CRT input
9 - CRT output
10 - user console input (valid for ISIS-II and ISIS-III (N) only)
11 - user console output (valid for ISIS-II and ISIS-III (N) only)
12 - teletype paper tape reader (valid for ISIS-II and ISIS-III (N) only)
13 - high speed paper tape punch (valid for ISIS-II and ISIS-III (N) only)
14 - user reader 1
15 - user reader 2
16 - teletype paper tape punch (teletype) (valid for ISIS-II and ISIS-III (N) only)
17 - high speed paper tape punch (valid for ISIS-II and ISIS-III (N) only)
18 - user punch 1 (valid for ISIS-II and ISIS-III (N) only)
19 - user punch 2 (valid for ISIS-II and ISIS-III (N) only)
20 - line printer
21 - user list 1 (valid for ISIS-II and ISIS-III (N) only)
22 - byte bucket (a pseudo input/output device)
23 - console input
24 - console output
25 - directory 6
26 - directory 7

27 - directory 8
28 - directory 9
29 - spool printer device
30 - batch input (Series IV SUBMIT file)
31 - batch output (Series IV LOG file)

The file name and extension are the ISIS file name.

The device type specifies the type of peripheral with which the file is associated. The possible values for this field are

0 - sequential input device
1 - sequential output device
2 - sequential input/output device
3 - random access input/output device

If the field is 3 the possible values for device are

0 - controller not present (valid for ISIS-II and ISIS-III (N) only)
1 - two-board double density (valid for ISIS-II and ISIS-III (N) only)
2 - two-board single density (valid for ISIS-II and ISIS-III (N) only)
3 - integrated single density (valid for ISIS-II and ISIS-III (N) only)
4 - reserved
5 - NDS-I remote mass storage (valid for NDS-I only)
6 - assigned device
7 - reserved
8 - unassigned device

Description

The SPATH call allows your program to obtain information relating to a specified file. The information returned by this call includes the device number, file name and extension, device type, and if a disk file, the drive type.

Examples

1. PL/M SPATH Call

```
SPATH:
 PROCEDURE (path$p, info$p, status$p) EXTERNAL;
    DECLARE (path$p, info$p, status$p) ADDRESS;
 END SPATH;
     .
     .
     .
    DECLARE FILENAM(15) BYTE;
    DECLARE FILINF STRUCTURE (DEVICE$ NO BYTE,
                             FILENAME (6) BYTE;
                             FILE$EXT (3) BYTE,
                             DEVICE$ TYPE BYTE,
                             DRIVE$ TYPE BYTE);
    DECLARE STATUS ADDRESS;
     .
     .
     .
    CALL SPATH (.(':F3:'),. INFO,.STATUS))
    IF STATUS<>0 THEN ...
    IF INFO.DRIVE$TYPE<>6
    THEN .../* UNASSIGNED*/
```

2. Assembly Language SPATH Call

```
        EXTRN   ISIS            ; LINK TO ISIS ENTRY POINT
SPATH   EQU     14              ; SYSTEM CALL IDENTIFIER
;
        MVI     C,SPATH         ; LOAD IDENTIFIER
        LXI     D,SBLK          ; LOAD PARAM ADDR
        CALL    ISIS
        LDA     SSTAT           ; TEST ERROR STATUS
        ORA     A
        JNZ     EXCEPT          ; BRANCH TO EXCEPTION
                                ; ROUTINE
.
.
.
SBLK;                           ; PARAMETER BLOCK FOR
                                ; SPATH
        DW      FILEN           ; POINTER TO FILE NAME
        DW      BUFIN           ; POINTER TO BUFFER
        DW      SSTAT           ; POINTER TO STATUS
;
FILEN:  DS      15              ; FILE NAME FIELD
BUFIN   DS      12              ; BUFFER FOR DATA
SSTAT:  DS      2               ; STATUS (RETURNED)
```

WRITE — Transfer Data from Memory to File

Syntax

```
CALL WRITE (conn, buf$p, count, status$p)
```

You must pass four parameters, *conn, buf$p, count,* and *status$p*, in the WRITE call,

where

conn	is the connection number (AFTN) of a file open for output or update. The AFTN was returned by a preceding OPEN call or is 0 for :CO:.
buf$p	is the address of the memory location of the buffer from which data is to be transferred, or a string literal of the format *.string literal*, where the period (.) specifies the contents of the string buffer labeled *.string literal.*
count	is the number of bytes to be transferred from the buffer to the output file. The value of the count is added to MARKER. If this results in MARKER being greater than LENGTH, then LENGTH is set equal to MARKER. The number of bytes actually transferred by WRITE is exactly equal to count. Thus, if the buffer length is less than count, memory locations following buffer are written to the file.
status$p	is the address of the memory location for the return of non-fatal error numbers. The error numbers returned by WRITE are listed in Appendix A.

Description

The WRITE call transfers data from a specified location in memory called a buffer to an open file.

Examples

1. **PL/M WRITE Call**

```
WRITE:
   PROCEDURE (conn, buf$p, count, status$p) EXTERNAL;
     DECLARE (conn, buf$p, count, status$p) ADDRESS;
   END WRITE;
.
.
.
DECLARE AFTOUT ADDRESS;
DECLARE BUFFER(128) BYTE;
DECLARE STATUS ADDRESS;
.
.
.
CALL WRITE (0,.('this is an example of string
literal', 0DH,0AH),38,.STATUS);
CALL WRITE (AFTOUT,.BUFFER,128,.STATUS);
IF STATUS<>0 THEN ...
.
.
.
```

2. Assembly Language WRITE Call

```
         EXTRN     ISIS          ; LINK TO ISIS ENTRY POINT
WRITE    EQU       4             ; SYSTEM CALL IDENTIFIER

         MVI       C,WRITE       ; LOAD IDENTIFIER
         LXI       D,WBLK        ; ADDRESS OF PARAMETER
                                 ; BLOCK
         CALL      ISIS
         LDA       WSTAT         ; TEST ERROR STATUS
         ORA       A
         JNZ       EXCEPT        ; BRANCH TO EXCEPTION
                                 ; ROUTINE
.
.
.
WBLK:                            ; PARAMETER BLOCK FOR
                                 ; WRITE
WAFT:    DS        2             ; FILE AFTN
         DW        OBUF          ; ADDRESS OF OUTPUT BUFFER
WCNT:    DW        128
         DW        WSTAT         ; POINTER TO STATUS
;
WSTAT:   DS        2             ; STATUS (RETURNED)
OBUF:    DS        128           ; OUTPUT BUFFER
;
```

Directory Maintenance

Five system calls are available to your program for changing and accessing information in a directory (see figure 3-3). These calls allow you to delete a file, rename a file, change the write-protect status of a file, and obtain directory information about a file.

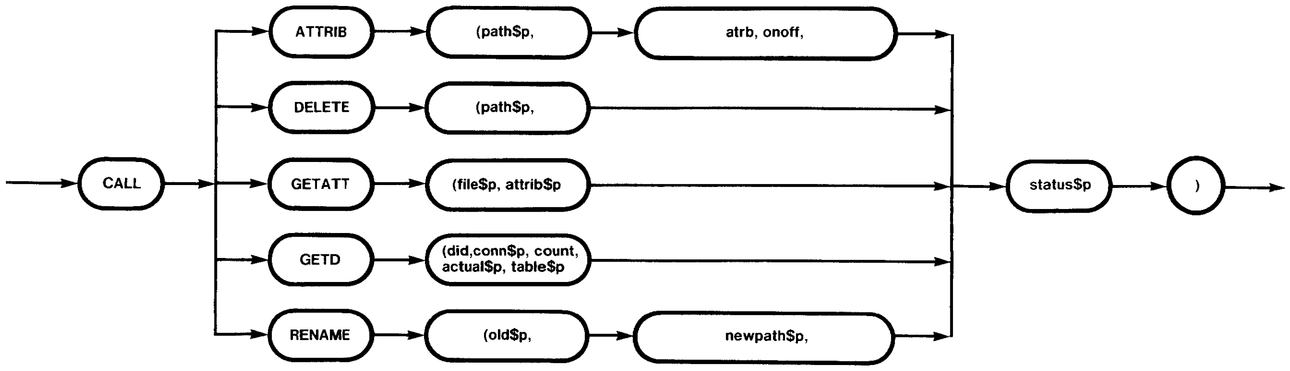


Figure 3-3. Directory Maintenance System Calls 121980-5

ATTRIB — Change the Write-Protect Status of a File

Syntax

`CALL ATTRIB` *(path$p, atrb, onoff, status$p)*

You must pass four parameters, *path$p, atrb, onoff,* and *status$p,* in the ATTRIB call,

where

path$p	is the address of an ASCII string containing the name of the file whose write-protect status is to be changed. The string can contain leading space characters but no embedded spaces. The string must be terminated by a character other than a letter, digit, colon (:), or period (.). A space can be used.
atrb	is a number indicating the attribute to be changed. The identifier can be 0 - invisible attribute (valid for ISIS-II and III (N) only) 1 - system attribute (valid for ISIS-II and III (N) only) 2 - write protect attribute. This attribute will set or reset the access right switches (Owner Write, Owner Delete, World Write, and World Delete) of the file. 3 - format attribute (valid for ISIS-II and III (N) only)
onoff	is a value indicating whether the attribute is to be set (turned on) or reset (turned off). The value is stored in the low order bit of the low order byte. A value of 1 specifies that the attribute be set and a value of 0 specifies that it be reset.
status$p	is the address of a memory location for the return of a non-fatal error number. The non-fatal error numbers issued by the ATTRIB call are listed in Appendix A.

Description

The ATTRIB call allows your program to change the write-protect status of a file. ATTRIB supports the ISIS write-protect attribute by mapping this attribute into the iNDX access rights (see table 3-3).

Table 3-3. Changing Write-Protect Status

Operation	Action
Set write protect on miniature flexible diskette	Deny WORLD WRITE Deny WORLD DELETE
Set write protect on hard disk or Winchester device	Deny WORLD WRITE Deny WORLD DELETE Deny OWNER WRITE Deny OWNER DELETE
Reset write protect on miniature flexible diskette	Grant WORLD WRITE Grant WORLD DELETE
Reset write protect on hard disk or Winchester device	Grant OWNER WRITE Grant OWNER DELETE

Examples

1. **PL/M ATTRIB Call**

```
ATTRIB:
 PROCEDURE (path$p, atrb, onoff, status$p) EXTERNAL;
    DECLARE (path$p, atrb, onoff, status$p) ADDRESS;
 END ATTRIB;
.
.
.
DECLARE FILE(15) BYTE;
DECLARE STATUS ADDRESS;
.
.
.
CALL ATTRIB (.FILE,2,0,.STATUS);
IF STATUS<>0 THEN ...
.
.
.
```

2. **Assembly Language ATTRIB Call**

```
        EXTRN   ISIS            ; LINK TO ISIS ENTRY POINT
ATTRIB  EQU     10              ; SYSTEM CALL IDENTIFIER
;
        MVI     C,ATTRIB        ; LOAD IDENTIFIER
        LXI     D,ABLK          ; LOAD PARAM ADDR
        CALL    ISIS
        LDA     ASTAT           ; TEST ERROR STATUS
        ORA     A
        JNZ     EXCEPT          ; BRANCH TO EXCEPTION
                                ; ROUTINE
;
ABLK:                           ; PARAMETER BLOCK FOR
                                ; ATTRIB
        DW      FILEN           ; POINTER TO FILE NAME
        DW      2               ; ATTRIBUTE IDENTIFIER
        DW      0               ; SET/RESET SWITCH
        DW      ASTAT           ; POINTER TO STATUS
;
FILEN:  DS      15              ; FILE NAME FIELD
ASTAT:  DS      2               ; STATUS (RETURNED)
;
```

DELETE — Delete a File from the Directory

Syntax

`CALL DELETE` *(path$p, status$p)*

You must pass two parameters, *path$p* and *status$p*, in the DELETE call,

where

path$p	is the address of an ASCII string that specifies the name of the file to be deleted. The file to be deleted must not be open. The string can contain leading space characters but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or period (.). You can use a space.
status$p	is the address of a memory location for the return of a non-fatal error number. The error numbers returned by DELETE are listed in Appendix A.

Description

The DELETE call removes a specified file from its directory. The file must not be open. The space allocated to the file is released. The space can then be reused for another file.

Examples

1. **PL/M DELETE Call**

```
DELETE:
 PROCEDURE (path$p, status$p) EXTERNAL;
    DECLARE (path$p, status$p) ADDRESS;
END DELETE;
.
.
.
DECLARE FILENAM(20) BYTE;
DECLARE STATUS ADDRESS;
.
.
.
CALL DELETE (.FILENAM,.STATUS);
IF STATUS<>0 THEN ...
.
.
.
```

2. Assembly Language DELETE Call

```
        EXTRN    ISIS          ; LINK TO ISIS ENTRY POINT
DELETE  EQU      2             ; SYSTEM CALL IDENTIFIER
;
        MVI      C,DELETE      ; LOAD IDENTIFIER
        LXI      D,DBLK        ; ADDRESS OF PARAMETER
                               ; BLOCK
        CALL     ISIS
        LDA      DSTAT         ; TEST ERROR STATUS
        ORA      A
        JNZ      EXCEPT        ; BRANCH TO EXCEPTION
                               ; ROUTINE
.
.
.
DBLK:                          ; PARAMETER BLOCK FOR
                               ; DELETE
        DW       DFILE         ; POINTER TO FILENAME
        DW       DSTAT         ; POINTER TO STATUS
;
DSTAT:  DS       2             ; STATUS (RETURNED)
DFILE:  DB       'FILE.EXT'    ; NAME OF FILE TO BE DELETED
;
```

GETATT — Obtain Write-Protect Information

Syntax

```
CALL GETATT (file$p, attrib$p, status$p)
```

You must pass three parameters, *file$p, attrib$p,* and *status$p,* in the GETATT call,

where

file$p	is the address of an ASCII string representing the name of the file for which attribute information is to be returned. The string can contain leading space characters, but no embedded spaces. It must end with a space.
attrib$p	is the address of a one-byte field to which the write-protect status of the file is to be returned. The following defines this byte: bit 0 set = invisible (valid for ISIS-II and III (N) only) bit 1 set = system (valid for ISIS-II and III (N) only) bit 2 set = write-protect bit 3 set = reserved bit 4 set = reserved bit 5 = reserved bit 6 = reserved bit 7 set = format (valid for ISIS-II and III (N) only)
status$p	is the address of an address variable for the return of the completion status for the requested operation. Zero indicates completion with no error. A non-zero value is a non-fatal error number. The error numbers issued by the GETATT system call are listed in Appendix A.

Description

The GETATT call allows your program to obtain the write-protect status of a file. GETATT can return only the write-protect attribute. The GETATT call returns the write-protect attribute under the following conditions:

- When the WORLD WRITE and DELETE access rights to the specified file have been denied.
- When the OWNER WRITE and DELETE access rights to the specified file have been denied, or the user is not the owner of the file.

NOTE

GETATT may return a false indication of the write-protect attribute if the user is the Superuser. The state of the Superuser's access rights does not effect the setting or resetting of the write-protect attribute to a file. Hence, the Superuser can have access to a file, yet GETATT could return a false write-protect indication.

Examples

1. PL/M-80 GETATT Call

```
GETATT:
 PROCEDURE (file$p, attrib$p, status$p) EXTERNAL;
  DECLARE(file$p, attrib$p, status$p)ADDRESS;
  END GETATT;
  /*MUST BE LINKED WITH SYSTEM.LIB*/
  .
  .
  .
  .
    DECLARE FILE(15) BYTE;
    DECLARE STATUS ADDRESS;
    DECLARE ATTRIB BYTE;
    .
    .
    .
    .
    CALL GETATT (.FILE, .ATTRIB, .STATUS);
    IF STATUS<>0 THEN ...
```

2. Assembly Language GETATT Call

```
        EXTRN   GETATT          ; MUST BE LINKED
                                ; WITH SYSTEM.LIB

;
; GETATT

        LHLD    FILEP           ; FILE PARAMETER
        PUSH    H
        LHLD    ATTP            ; ATTRIBUTE POINTER
        PUSH    H
        POP     B
        LHLD    STATP           ; STATUS
        PUSH    H
        POP     D
        CALL    GETATT
        LDA     GSTAT           ; TEST ERROR STATUS
        ORA     A

,...    JNZ     EXCEPT          ; BRANCH TO EXCEPTION
                                ; ROUTINE
FILEP:  DW      GFILE           ; POINTER TO FILE
                                ; PATHNAME
ATTP:   DW      ATTRIB          ; POINTER TO ATTRIBUTE

STATP:  DW      GSTAT           ; POINTER TO STATUS

;
GFILE:  DB      ':F0:FILE.EXT'  ; FILE PATHNAME

ATTRIB: DS      1               ; ATTRIBUTE VALUE
                                ; (RETURNED)

GSTAT:  DS      2               ; STATUS (RETURNED)
```

GETD — Obtain File Device Directory Information

Syntax

```
CALL GETD (did, conn$p, count, actual$p, table$p, status$p)
```

You must pass six parameters, *did, conn$p, count, actual$p, table$p,* and *status$p,* in the GETD call,

where

did	is the address variable containing the number for the directory from which entries are to be returned. Valid integer values are 0-9, representing :F0:-:F9:.
conn$p	is an address value that must be initialized to zero when the first display request of a sequence is made. Changing this value will cause an error. The system assigns a value to this word that is returned in subsequent requests for directory entries. This value acts as a pseudo-connection for subsequent calls.
count	is an address value that contains the number of entries to be returned. A value of 0 terminates the current display request sequence and releases the pseudo-connection. The program must make a final call to GETD with count=0 to release the connection unless it is automatically released.
actual$p	is an address value containing the number of directories returned by the system. When *actual* is less than *count*, the last directory entry has already been returned and the pseudo-connection will be released automatically.
table$p	is the address of a memory structure where directory entries are returned. The structure form is

```
DECLARE ENTRY STRUCTURE(

   RESERVED1 (1) BYTE,
   FILE$NAME (9) BYTE,
   RESERVED2 (6) BYTE);
```

where

FILE$NAME	is a 9-byte field with two subfields left-justified and zero-filled. The first 6 bytes represent the name and the remaining 3 bytes are the file extension.
status$p	is a pointer to the memory location for the return of the status for the requested operation. Zero indicates no error. A nonzero value corresponds to a non-fatal error number. The error numbers issued by the GETD system call are listed in Appendix A.

Description

The GETD call allows your program to access information in a file device directory defined by DID.

CAUTION

You must load ISIS overlay (ISIS.OV0) into the top of a 64K memory space with the LOAD call before calling GETD. The ISIS overlay will reside between 0E800H and 0F6BFH. Access the overlay entry point by linking to SYSTEM.LIB. Do not overwrite the overlay when accessing the directory desired. Reserved fields returned by GETD are different from one ISIS operating system to another. Do not use reserved fields in a program that employs GETD.

Examples

1. **PL/M-80 GETD Call**

```
GETD:
 PROCEDURE (did, conn$p, count, actual$p, table$p, status$p) EXTERNAL;
  DECLARE (did, conn$p, count, actual$p, table$p, status$p) ADDRESS;

 END GETD;

 DECLARE (dummy,status,conn,actual) ADDRESS;

 DECLARE TABLE (50) STRUCTURE
                (Reserved (1) Byte,
                 Filename (9) Byte,
                 Reserved (6) Byte) ;

 CONN=0;

 CALL LOAD (.(':F0:ISIS.OV0'),0,0, .DUMMY,.STATUS);

 CALL GETD (1,.CONN, 50, .ACTUAL, .TABLE (0), .STATUS);

 IF STATUS<>0 THEN...
```

2. Assembly Language GETD Call

```
;
ETD
        EXTRN    GETD          ; MUST BE LINKED
                               ; WITH SYSTEM.LIB
UIS.OVO MUST BE LOADED BEFORE CALLING GETD
        LXI      H,0
        SHLD     CONNP         ; INITIAL 0
        LHLD     DID           ; DIRECTORY
        PUSH     H
        LXI      H,CONNP       ; CONNECTION POINTER
        PUSH     H
        LHLD     COUNT         ; COUNT
        PUSH     H

        LHLD     ACTP          ; ACTUAL
        PUSH     H

        LHLD     BUFFP         ; BUFFER POINTER
        PUSH     H
        POP      B
        LHLD     GSTAT         ; STATUS
        PUSH     H
        POP      D

        CALL     GETD
        LDA      DSTAT         ; TEST ERROR STATUS
        ORA      A
        JNZ      EXCEPT        ; BRANCH TO EXCEPTION
                               ; ROUTINE
;
;
DID     DW       1             ; DIRECTORY IDENTIFIER
CONNP:  DW       0             ; DIRECTORY CONNECTION
COUNT:  DW       8             ; ENTRY COUNT
ACTP:   DW       DACT          ; POINTER TO ACTUAL
BUFFP:  DW       DBUF          ; POINTER TO BUFFER
GSTAT:  DW       DSTAT         ; POINTER TO STATUS
DACT:   DS       2             ; COUNT OF ENTRIES READ
                               ; (RETURNED)
DBUF:   DS       128           ; DIRECTORY BUFFER
DSTAT:  DS       2             ; STATUS (RETURNED)
;
```

Console Device Assignment and Error Message Output

Three system calls are available to your program for system console control (see figure 3-4). These system calls allow you to change the device used as the system console, to determine which device is the current console, and, if needed, to send an error message to the console.

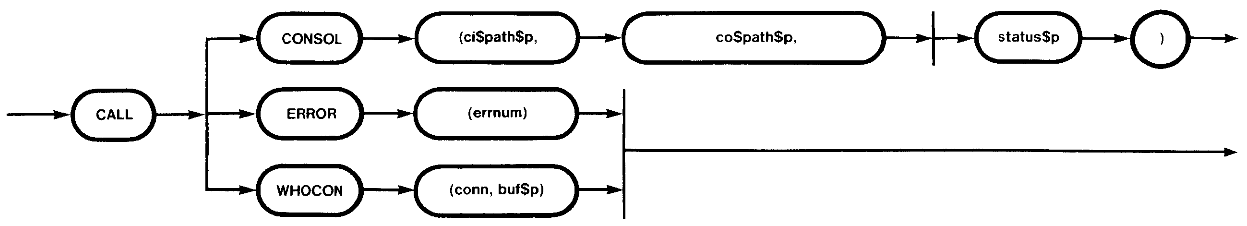


Figure 3-4. Console Control System Calls 121980-6

CONSOL — Change Console Device

Syntax

`CALL CONSOL` *(ci$path$p, co$path$p, status$p)*

You must pass three parameters, *ci$path$p, co$path$p,* and *status$p,* in the CONSOL call,

where

ci$path$p	is the address of an ASCII string that contains the name of the file to be used for the system console input. The string can contain leading spaces but no embedded spaces. It must end with a character other than a letter, digit, colon (:), or period (.). You can use a space. If :CI: is the specified file, the console input will not be changed. If the specified file is not :CI:, the old console input file will be closed and the specified file (e.g., :BI:) will be opened. Do not use SUBMIT to run programs that contain CONSOL calls that change the console input device name.
co$path$p	is the address of an ASCII string that contains the name of the file to be used for system console output. The string can contain leading spaces but no embedded spaces. It must end with a character other than a letter, digit, colon (:) or period (.). You can use a space. If :CO: is the specified file, the console output will not be changed. If the specified file is not :CO:, the old console output file will be closed, and the specified file (e.g., :BO:) will be opened.
status$p	is the address of a memory location. *status$p* does not return any non-fatal errors.

Description

The CONSOL call allows your program to change the console input and output devices (:CI: and :CO:) to devices other than the initial system console.

Possible Error Conditions

A fatal error will occur when

- The specified console input file cannot be opened (e.g., opening :VI: when you are executing ISIS-IV under background or import; opening :BI: when you have interactively invoked ISIS-IV).
- The specified console output file cannot be opened (e.g., opening :VO: when you are executing ISIS-IV under background or import; opening :BO: when you have no Series IV LOG file at the time that ISIS-IV is invoked).

Examples

1. PL/M CONSOL Call

```
CONSOL;
 PROCEDURE (ci$path$p, co$path$p, status$p) EXTERNAL;
    DECLARE (ci$path$p, co$path$p, status$p) ADDRESS;
 END CONSOLE;
.
.
.
DECLARE INFILE(6) BYTE;
DECLARE OUTFILE(6) BYTE;
DECLARE STATUS ADDRESS;
.
.
.
CALL CONSOL (.INFILE,.OUTFILE,.STATUS);
IF STATUS<>0 THEN...
.
.
.
```

2. Assembly Language CONSOL Call

```
        EXTRN    ISIS          ; LINK TO ISIS ENTRY POINT
CONSOL  EQU      8             ; SYSTEM CALL IDENTIFIER
;
        MVI      C,CONSOL      ; LOAD IDENTIFIER
        LXI      D,CBLK        ; LOAD PARAM ADDR
        CALL     ISIS
        LDA      CSTAT         ; TEST ERROR STATUS
        ORA      A
        JNZ      EXCEPT        ; BRANCH TO EXCEPTION
                               ; ROUTINE
;
CBLK:                          ; PARAMETER BLOCK FOR
                               ; CONSOL
        DW       INFILE        ; POINTER TO FILE NAME
        DW       OTFILE        ; POINTER TO FILE NAME
        DW       CSTAT         ; POINTER TO STATUS
;
INFILE: DS       15            ; INPUT FILE NAME
OTFILE: DS       15            ; OUTPUT FILE NAME
CSTAT:  DS       2             ; STATUS (RETURNED)
;
```

ERROR — Output Error Message on System Console

Syntax

```
CALL ERROR (errnum)
```

In assembly language you must pass two parameters, *errnum* and the address of a memory location for return of an error number, in the ERROR call. You must pass only one parameter, *errnum*, in PL/M,

where

errnum	is the error number. The error number must be in the low order eight bits of the parameter. The high order eight bits are ignored. Only the numbers 101 through 199 inclusive should be used for the user programs; the other numbers (0-100 and 200-255) are reserved for system programs. The format of the rror is ERROR *dd* USER PC *hhhh*.

where

dd	is the decimal number of the error specified in the call.
hhhh	is the hexadecimal number of the return address in the calling program. For error numbers in the range of 0E100H through 0E1FFH ERROR reports the DQ$DECODE$EXCEPTION message.

Description

The ERROR call enables your program to report an error message. In foreground, but not import, the error is sent to the screen. In background or import the error is sent to the ISIS file :CO:.

Examples

1. PL/M ERROR Call

```
ERROR:
 PROCEDURE (errnum) EXTERNAL;
    DECLARE (errnum) ADDRESS;
 END ERROR;
.
.
.
DECLARE ENUM ADDRESS;
.
.
.
CALL ERROR (ENUM);
.
.
.
```

2. Assembly Language ERROR Call

```
        EXTRN   ISIS
ERROR   EQU     12              ; CALL IDENTIFIER
;
        MVI     C,ERROR         ; LOAD IDENTIFIER
        LXI     D,EBLK          ; LOAD PARAM ADDR
        CALL    ISIS
;
EBLK:
ERNUM:  DS      2               ; ERROR NUMBER FIELD
        DW      STATUS          ; ISIS-IV WANT TO RETURN A
STATUS: DS      2               ; STATUS, SO PUT IT HERE
```

WHOCON — Determine File Assigned as System Console

Syntax

CALL WHOCON (*conn, buf$p*)

In assembly language you must pass three parameters, *conn, buf$p*, and the address of a memory location for return of an error number, in the WHOCON call. You must pass only two parameters, *conn* and *buf$p*, in PL/M,

where

conn	is a value that indicates whether the input or output file (:CI: or :CO:) name is to be returned. A value of 0 specifies output and a value of 1 specifies input.
buf$p	is the address of a 15-byte buffer reserved by your program for the return of the name of the file assigned to :CI: or :CO:. The name is returned as an ASCII string terminated by a space.

Description

The WHOCON call allows your program to determine what file is assigned as the current system input console or output console.

Examples

1. PL/M WHOCON Call

```
WHOCON:
 PROCEDURE (conn, buf$p) EXTERNAL;
    DECLARE (conn, buf$p) ADDRESS;
 END WHOCON;
.
.
.
DECLARE BUFF$IN(15) BYTE;
.
.
.
CALL WHOCON(1,.BUFF$IN);
.
.
.
```

2. Assembly Language WHOCON Call

```
        EXTRN   ISIS
WHOCON  EQU     13              ; CALL IDENTIFIER
;
        MVI     C,WHOCON        ; LOAD IDENTIFIER
        LXI     D,WBLK          ; LOAD PARAM ADDR
        CALL    ISIS
;
WBLK:
AFTN:   DS      2               ; AFTN FOR IN OR OUT
        DW      BUFIN           ; POINTER TO BUFFER
        DW      STATUS          ; POINTER TO STATUS RETURN
;
BUFIN:  DS      15              ; BUFFER FOR RETURN
                                ; FILE NAME
STATUS: DS      2               ; STATUS RETURN
;
```

Program Execution

Two system calls allow your program to transfer control to another program (LOAD) or to ISIS (EXIT) (see figure 3-5). The LOAD call can be used to load another program and then transfer control to it, to the Monitor, or have control returned to the calling program. The EXIT call is used to terminate processing and return to ISIS.

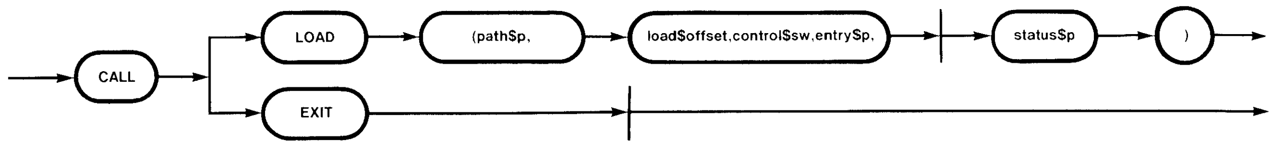


Figure 3-5. Program Execution System Calls 121980-7

EXIT — Terminate the Program and Return to ISIS-IV

Syntax

```
CALL EXIT
```

You pass no parameters in a PL/M call to EXIT. In an assembly language call, one parameter is passed: the address of a memory location for return of an error number.

Description

The EXIT call can be used by your program to terminate execution and return to ISIS-IV. All open files are closed, with the exception of :CO: and :CI:. The current system console assignment is not changed.

Examples

1. **PL/M EXIT Call**

```
EXIT:
 PROCEDURE EXTERNAL;
 END EXIT;
.
.
.
CALL EXIT;
.
.
.
```

2. **Assembly Language EXIT Call**

```
        EXTRN   ISIS
EXIT    EQU     9           ; CALL IDENTIFIER
;
        MVI     C,EXIT      ; LOAD IDENTIFIER
        LXI     D,EBLK      ; LOAD PARAM ADDR
        CALL    ISIS
;
EBLK:
        DW      ESTAT       ; POINTER TO STATUS
;
ESTAT:  DS      2           ; STATUS FIELD
;
```

LOAD — Load a File of Executable Code and Transfer Control

Syntax

`CALL LOAD` *(path$p, load$offset, control$sw, entry$p, status$p)*

You must pass five parameters, *path$p, load$offset, control$sw, entry$p*, and *status$p*, in the LOAD call,

where

path$p	is the address of an ASCII string containing the name of the file to be loaded. The string can contain leading spaces but no embedded spaces. It must be terminated by a character other than a letter, digit, colon (:), or a period (.). You can use a space.
load$offset	is a bias value to be added to the load address of the program. The program is loaded at the adjusted address. The use of the bias does not mean that thc program is relocatable. Usually the code cannot be executed at the biased addrcss. For most applications, the bias will be zero.
control$sw	is a value indicating where control is transferred after the load. A value of zero returns control to the calling program. The debug toggle is unchanged. A value of one transfers control to the load program. The debug toggle is reset. If the program is not a main program, its entry point is zero, which causes control to vector through location zero to the Monitor. A value of two transfers control to the Monitor. The debug toggle is set. The Monitor Execute (G) command can be used to start the program.
entry$p	is the address of a memory location for the return of the loaded program entry point address when the control value is zero. The entry point is obtained from the loaded program. A zero is returned if the program is not a main program.
status$p	is the address of a memory location for the return of a non-fatal error number. The error numbers issued by the LOAD call are listed in Appendix A.

Description

The LOAD call allows your program to load a located or absolute object file. After the file is loaded, control is passed to the loaded program, the calling program, or to the Monitor depending on the value of a parameter.

Examples

1. PL/M LOAD Call

```
LOAD:
 PROCEDURE (path$p, load$offset, control$sw, entry$p, status$p) EXTERNAL;
    DECLARE (path$p, load$offset, control$sw, entry$p, status$p) ADDRESS;
 END LOAD;
.
.
.
DECLARE FILNAM(15) BYTE;
DECLARE ENTRY ADDRESS;
DECLARE STATUS ADDRESS;
.
.
.
CALL LOAD (.FILNAM,0,1,.ENTRY,.STATUS);
IF STATUS<>0 THEN ...
.
.
.
```

2. Assembly Language LOAD Call

```
        EXTRN   ISIS
LOAD    EQU     6               ; CALL IDENTIFIER
;
        MVI     C,LOAD          ; LOAD IDENTIFIER
        LXI     D,LBLK          ; LOAD PARAM ADDR
        CALL    ISIS
        LDA     LSTAT           ; TEST ERROR STATUS
        ORA     A
        JNZ     EXCEPT          ; BRANCH IF ERROR
;
LBLK
        DW      FILNAM          ; POINTER TO FILE NAME
BIAS:   DS      2               ; BIAS FIELD
SWITCH: DS      2               ; CONTROL SWITCH
        DW      ENAD            ; POINTER TO ENTRY ADDRESS
        DW      LSTAT           ; POINTER TO STATUS
;
FILNAM: DS      15              ; FILE NAME FIELD
ENAD:   DS      2               ; ENTRY POINT ADDR (RETURN)
LSTAT:  DS      2               ; STATUS (RETURNED)
;
```

CHAPTER 4
MON 85

- MON 85 is provided for 8080/8085 applications program debugging.
- Deb 88 is provided for 8086/8088 applications program debugging. Refer to the *Intellec Series IV Operating and Programming Guide*, 121753.

Monitor Usage and Activation

The ISIS Execution Unit Monitor (MON 85) provides you with the basic utility functions for debugging 8080/8085-based programs. MON 85 allows you to

- Execute, single step, and breakpoint a program
- Display, modify, and scan memory
- Display and modify CPU registers
- Input from and output to I/O ports
- Perform arithmetic operations
- Disassemble instructions

MON 85 can be entered by

- Typing Debug at the ISIS prompt (-)
- Pressing function key 0
- Transferring control to location 0 in programs
- Giving the ISIS-IV LOAD system-call the value of 2
- Executing a program without a start-address
- A fatal system error in debug mode

Command Line Editing

MON 85 provides you with line-oriented command line editing capabilities. Refer to table 3-2 for more details on editing keys.

Entering Monitor Commands

You can enter commands at the console any time after the monitor prompt character, a period (.), is displayed at the left margin.

Monitor commands are single alphabetic characters. Some of these commands have optional parameters. For example, the Examine Command (X) displays the contents of all registers. If you want to see the contents of only one register, you must specify that register in the command.

Other commands have required parameters. For example, the Display Command (D), which displays the contents of memory, requires that beginning and ending memory locations of a memory block be given in the command.

Some commands have optional characters that modify command functions. For example, the D has an optional X suffix that allows you to display memory in disassembled code.

Normally, commands are terminated by pressing carriage return on the keyboard. There are some exceptions to this that will be explained in the individual command descriptions.

Command Syntax

The general command syntax of MON 85 commands is

command [*parameters*] < c r >

where

command	is the single alphabetic character for the basic command.
parameters	are one or more variable data supplied with the command.

Parameters can be expressions, memory addresses, registers, and ranges. Numeric fields within these parameters are entered as either hexadecimal digits (in the range of 0000-FFFF) or decimal digits. A decimal digit generally requires the suffix T to distinguish it from hexadecimal except when used as a repeat or count factor.

The memory addresses and range specifiers are flexible in that numbers and registers can be added to or subtracted from one another, thereby evaluating to a hexadecimal number. The following list defines these parameters:

```
dec digit::= {0|1|2|3|4|5|6|7|8|9}
hex digit::= { <dec digit> |A|B|C|D|E|F}
dec number::= { <dec digit> <dec number> | <dec digit> }
hex number::= { <hex digit> <hex number> | <hex digit> }
number::= <dec number> | <hex number>
add reg::= {RB|RD|RH|PC|SP}
byte reg::= {RA|RB|RC|RD|RE|RH|RL|RF}
reg::= {RA|RB|RC|RD|RE|RH|RL|PC|SP|RF}
addr expr::= { <addr term> | <addr expr> { + | - }
      <addr term> }
addr term::= { <number| <addr reg> }
byte expr::= { <byte term> | <byte expr> { + | - } <byte term> }
byte term::= { <number> | <byte reg> }
addr::= {[+]<addr expr> }
```

Memory Addresses

Memory addresses used with the MON 85 consist of one to four digits. Longer numbers can be entered but only the four rightmost digits are used by the system. For example, the value 123456 hexadecimal is treated as 3456 hexadecimal. Hexadecimal byte values consist of one or two digits. Longer hexadecimal byte values can be entered, but only the two rightmost digits are used.

The types of memory addresses used in MON 85 commands are

- *start-address* — the address where a display begins or where the control of a system is passed
- *destination-address* — the beginning address of a block of memory that is going to be compared to another block of memory, or the address where a block of memory is moved
- *break-address* — address in a program where control is passed back to the monitor

CAUTION

With MON 85, you can examine and modify memory addresses 3180 hexadecimal through 0F6C0 hexadecimal inclusive, and interrupt vectors 3 through 7 (18 hexadecimal through 3F hexadecimal) unless they are being used by an Intel product. You should not modify the memory outside this range or ISIS and the monitor may malfunction and your data and files may be destroyed.

Registers

A register evaluates its contents when the command referencing it is executed. Registers are referenced in command syntax by two letter mnemonics. Table 4-1 lists register mnemonics for the MON 85.

Table 4-1. Register Mnemonics for MON 85

Register Name	Mnemonic
CPU A Register	RA
CPU B Register	RB
CPU C Register	RC
CPU D Register	RD
CPU E Register	RE
CPU H Register	RH
CPU L Register	RL
Program Counter	PC
Stack Pointer	SP
Flag	RF

Repeat Factors

There are two ways to put more than one command on one command line. First, you can put separate commands in the same command line if your separate them with semicolons. Second, you can specify that a command be repeated any desired number of times if you enclose the command in angle brackets and place a decimal repetition factor ahead of the first bracket. A repetition factor of *n* says, "Do this command *n* times." For example,

```
5<12<G, 3B7>; D 4A>
```

is a valid command line in which the Execution Command (G) is repeated 60 times and the Display Command (D) is repeated 5 times. Note that the T suffix is not used for the repeat factors since only decimal numbers are accepted.

Count Factors

Count factors are closely related to repetition factors. When you put a decimal count factor immediately ahead of a command letter or letters, you are telling the monitor, "Do a command for *n* items at a time." For example,

```
20D 1B
```

causes the display of 20 consecutive bytes beginning at address 1B. Note that a T suffix is not used since only decimal numbers are accepted.

Range Specifications

A monitor command can act on a block of memory called a range. You can specify the range in one of the following forms:

- A start-address followed by an end address and separated by an exclamation mark (!) (e.g., 1234!5678)
- A start-address followed by a length in bytes and separated by a pound sign (#) (e.g., 1234#5F)

Note that *range*, used in the syntax descriptions of the commands, refers to a start address followed by either an exclamation mark and an end-address or a pound sign and a length in bytes.

Also remember this additional information about MON 85 range specifications.

- If you omit the start-address, PC is assumed.
- When a range is expected and you don't specify an end-address or length, the range is taken to be a single byte.
- PC is added to the address when the address begins with a plus sign (+).

Command-Line Commenting

You can enter comments into command-lines by typing an asterisk followed by a comment. To put a command and a comment on the same command-line, enter a semicolon following the command and then type the asterisk and the comment.

Example

```
.*This whole line is a comment
.2D 10E; *This line ends with a comment
```

Error Messages

If you enter an invalid command, the monitor notifies you with one of the following error messages:

Syntax Error
Bad Command
Invalid Expression
Improper nesting of repeat factors

Remember that when an error is detected in a command-line composed of multiple commands, all commands prior to the error will be executed.

Monitor Commands

Categories of Monitor commands are grouped as follows:

- Program execution commands

Execute (G)	Begins program execution and provides breakpoints

Single Step (N)	Displays and executes a specified number of instructions
• Memory control commands	
Compare (C)	Compares contents of one block of memory to contents of another block of memory
Display (D)	Displays contents of a specified block of memory in byte, word, or instruction format
Find (F)	Searches a block of memory for a sequence of hexadecimal digits
Move (M)	Moves a block of memory to another location in memory
Substitute (S)	Dispays one or more memory locations and optionally modifies them on a byte or word basis
• Register command	
Examine Register (X)	Displays and optionally modifies register contents
• Utility command	
Print Value Command (P)	Displays a literal string, an address, or the value of an expression

See Appendix C for a summary of the command syntax of these Monitor commands.

Program Execution Commands

The program execution commands, Execute and Single Step, are described in detail on the following pages.

G — Execute

Syntax

G[*start-address*] [, { *break-address* }] < c r >

Description

The Execute command transfers control to a program at the address you specify in the command and optionally sets one to four breakpoints in the program to which control is passed.

The start-address is optional. If you don't specify it in the MON 85 command, execution will begin at PC. You may enter this start-address directly following the G. Break-addresses and ranges 0 to FFFF, however, must be separated from start-address and G with a comma.

Breakpoints. You can set execution breakpoints in MON85 G commands. An execution breakpoint occurs when a breakpoint address is fetched. To set an execution breakpoint, enter the G command followed by either a start-address, a comma, and a break-address, or a comma and the break-address.

The instruction at the break-address is not executed until control is returned to the program. When an execution breakpoint occurs, the monitor prints

BREAK at *hex address*

and issues the period prompt (.).

NOTE

When setting an execution breakpoint, be sure that there is enough memory for six words to be pushed on the stack. You can ensure this by allocating 12 extra bytes to the stack.

If you would like to execute the G command successively with the same breakpoint (as in doing loop iterations), enter the G, PC command. This causes the start-address to be executed once without breaking and a break to occur on the next reference.

Examples

```
.G<cr>
.G , PC<cr>
.10<G40,40;><cr>
```

In the first example, execution begins with the instruction at the address given by PC.

In the second example, execution begins at PC and continues until the instruction at PC is about to be fetched again. The instruction at which it stops is fetched, but not executed.

In the third example, execution begins at 40 and executes 40 ten times. The register is display ten times.

N — Single Step

Syntax

[*count*] N [P] [start-address] [,] < c r >

Description

The Single Step command allows you to display and then execute one or more instructions. The decimal count factor specifies the number of instructions to be displayed and executed. All but the last instruction displayed are executed. For example, if you want to execute 23 instructions, you must specify the count factor as 24. This executes 23 instructions and displays the twenty-fourth. If you do not specify a count factor, the default is one.

If you include a start-address in the MON 85 command line, PC is modified and execution begins at this start-address.

When a start-address for the MON 85 command is not specified, execution begins at the locations indicated by the PC register.

Single Step Continuation. If you want to continue in single step mode, insert a comma at the end of the command line and then execute the command. The specified number of instructions is displayed followed by a special hyphen prompt (-). The monitor then waits for another comma. When you enter this comma, the displayed instructions are executed, the next instructions are displayed, and the monitor issues another prompt or a comma. Repeat this cycle as often as you desire. When finished, return to the general command level by entering a carriage return.

Command Suffix. A P command suffix is available if you are using MON 85. This optional command causes any routine called by the CALL or conditional CALL instruction to be treated as a single instruction. The CALL instruction is displayed, the routine is executed, and the next instruction after the routine is displayed.

Examples

```
.N,<cr>
.24N 4,<cr>
```

In the first example, the current instruction is displayed and the monitor waits for your input. If you type a comma, the displayed instruction is executed and the process repeats. If you type a carriage return, you return to the general command level.

In the second example, execution and display begin at 4 and continue until 23 instructions are executed. The twenty-fourth instruction is displayed but not executed, and the monitor waits for either the continuation comma or a carriage return.

Memory Control Commands

The memory control commands, Compare, Display, Find, Move, and Substitute, are described in detail on the following pages.

In the second example, sixteen bytes, starting with the byte at 3E0, are displayed in both hexadecimal and ASCII, and then a hyphen prompt is given. A comma is entered after the hyphen prompt and the next sixteen bytes are displayed followed by another hyphen prompt. A carriage return is then entered, terminating the command.

In the third example, three words in hexadecimal starting with SP are displayed.

F — Find

Syntax

F *range* , *data*<cr>

Description

The Find command allows you to search a block of memory for a sequence of hexadecimal digits. The block you indicate by range is searched for the sequence of digits you put in your data argument. Each time a match is found, the address of the first matching byte is displayed.

Your data sequence can contain from 1 to 32 hexadecimal digits representing from 1 to 16 bytes. An odd number of digits is not allowed except when there is only one digit in your data sequence. When there is only one digit in your data sequence, it is treated as if it had been entered with a leading zero.

Example

```
.F 18!PC, 54455354<cr>

0118
01A4
0212
```

In this example, the ASCII string TEST is found in three locations between the start of the code segment and the first instruction. In each case, the first address of the string is displayed.

M — Move

Syntax

M *range*, *destination-address*<cr>

Description

The Move command moves a block of memory to another location in memory. The block defined by range is moved to the location beginning at destination-address.

Example

```
.M 32CD #15T,404A<cr>
```

This command-line moves the 15 bytes beginning at 32CD to the location starting with 404A.

S — Substitute

Syntax

[count] S [W][*start-address*][=*expression*][| *expression*]...[,]<cr>

Description

The Substitute command allows you to display one or more memory locations and gives you the option of modifying each location on a byte or word basis. The optional count factor allows you to indicate the number of bytes or words to be displayed each time a display is requested. If you include a start-address, display begins there. If you do not, the display begins at the locations indicated by the PC register for MON 85.

Optional Command Suffix. You can use an optional W in MON 85. The W causes the displays resulting from the Substitute command to be on a word basis. When using the S command with the optional W, enter address expressions. When using the S command without the optional W, enter byte expressions.

Modifying Memory. You can modify memory with Substitute commands in several ways. You can enter S, use the optional start-address if desired, and then enter a carriage return. This displays the desired memory locations followed by a hyphen prompt (-). You can then enter new values or terminate the command with a carriage return.

A second way to modify memory using the Substitute Command is by entering S, skipping the display portion of the command, and then entering the memory location, an equal sign, and then the new contents. If you are modifying a block of memory, you must enter the new expressions following the equal sign or hyphen and separate them with a slash. This causes the indicated expressions to be placed in memory beginning at the first location displayed.

Substitute Continuation. If you want to continue in substitute mode, insert a comma at the end of the command line and then execute the command. The new values are placed in memory, the next location is displayed on the next line, and a hyphen prompt is issued. You can then enter more expressions. You can repeat this cycle as often as you desire. When finished, you can return to the general command level by typing a carriage return.

Examples

```
.S<cr>

0074  40  - 1/2/3,
0077  20  - 4/15/<cr>

.SW  3234 = FFFF/FFFF/FFFF<cr>

.4SW  4044 cr

4044  0001 0002 0003 0004 - 9090,
4048  0005 0006 0007 0008 - 9090<cr>
```

In the first example, the current location is displayed, the prompt is issued, and the values 1, 2, and 3 are placed in locations 74 through 76. The comma indicates continue, so the next value is displayed, another prompt is issued, and new values are put in locations 77 and 78. The carriage return terminates the process.

In the second example, there is no display and FFFF is placed in three consecutive words of memory, beginning at location 3234.

In the third example, four words are displayed and 9090 is placed in location 4044. The comma indicates continue, so the next four words are displayed and 9090 is placed in location 4048. The carriage return terminates the process.

Register Command

The register command, Examine Register, is described in detail on the following page.

X — Examine Register

Syntax

```
X [register {[=expression]|, }]<cr>
```

Description

The Examine register command allows you to examine and optionally modify register contents. It functions in several ways.

You can display all registers and their contents by just typing X. If you want to examine one register at a time, you can enter the X followed by the mnemonic of the register you want to see. This causes the register mnemonic and that register's contents to be displayed followed by a hyphen prompt (-). If you want to modify the contents of the register, you enter the new hexadecimal value after the hyphen. Typing a comma (,) will display the next register; otherwise, you type a carriage return.

If you want to modify a register without a display and a hyphen, you can enter the command followed by the register mnemonic, an equal sign, and then the expression. The modification is then immediate.

To modify the SP and PC registers in the MON 85 system, an address expression must be given. For all other registers, a byte expression must be given.

Examples

```
.X<cr>
.X  RA
.X  =  3B  -<cr>
.X  RA  =  RA  +  2F<cr>
```

In the first example, all registers are displayed.

In the second example, the RA register is displayed and a hyphen prompt (-) for a new hexadecimal value or a carriage return is issued. In the example, a carriage return is entered, terminating the command. Using a comma instead of a carriage return displays the next register along with new hyphen prompt.

In the third example, the RA register is set to its previous value plus the quantity 02FD and nothing is displayed.

Utility Command

The utility command, Print Value, is described in detail on the following page.

Monitor I/O Interface Routines

CI — Console Input Routine

The Console Input Routine reads a character entered at the keyboard and returns the character as a byte value if called from PL/M or in the RA register if called from the assembler. RA and PC registers and CPU condition codes are affected. If ISIS-II is executing in the background or in the import mode, calling CI will cause a fatal error that will cancel the user's job.

1. **PL/M CI Call Example**

   This is an example of a routine that reads a string of characters from the keyboard. The routine terminates when a carriage return is detected or when the number of characters specified by BUFSIZ has been read. If a carriage return is detected, the DONE code is executed, and if the buffer is filled, the OVFL code is executed.

```
CI: PROCEDURE BYTE EXTERNAL;      /*ENTRY POINT INTO SYSTEM.LIB*/
  END CI;

DECLARE BUFSIZ LITERALLY '122';   /*BUFFER SIZE*/
DECLARE BUFFER(BUFSIZ) BYTE;      /*BUFFER FOR STORING CHARACTERS*/
DECLARE INDEX BYTE;               /*INDEX INTO BUFFER*/
DECLARE CR LITERALLY '0DH';       /*CARRIAGE RETURN*/

INDEX = 0;
BUFFER(INDEX) = CI AND 7H;        /*READ IN CHARACTER AND STRIP OFF*/
                                  /*PARITY BIT*/
DO WHILE BUFFER(INDEX)<> CR;
 IF INDEX < LAST (BUFFER);
  DO;
   INDEX = INDEX + 1;
   BUFFER(INDEX) = CI AND 7FH;    /*CONTINUE READING UNTIL A
                                  /*CARRIAGE RETURN HAS BEEN INPUT*/
                                  /*OR THE BUFFER IS FULL*/
  END;
 ELSE
  DO;
   /*OVFL CODE*/
   END;
END;
/*DONE CODE*/
```

2. Assembly Language CI Call Example

```
          EXTRN   CI          ; ENTRY POINT INTO SYSTEM.LIB
                              ; FOR CI
BUFSIZ    EQU     122         ; BUFFER SIZE
CR        EQU     0DH         ; CARRIAGE RETURN
BUFFER:   DS      BUFSIZ      ; BUFFER
;
          LXI     H,BUFFER    ; HL POINT TO BEGINNING OF
                              ; BUFFER
LOOP:
          CALL    CI          ; GET CHARACTER
          ANI     7FH         ; STRIP OFF PARITY
          MOV     M,A         ; STORE IT IN BUFFER
          CPI     CR          ; IS IT A CARRIAGE RETURN
          JZ      DONE        ; IF IT IS, JUMP TO THE DONE
                              ; CODE
          INX     H           ; OTHERWISE, MOVE THE
                              ; BUFFER POINTER
          DCR     D           ; DECREASE CHARACTER
                              ; COUNT
          JZ      OVFL        ; IF BUFFER FULL, JUMP TO THE
                              ; OVFL CODE
          JMP     LOOP        ; GET THE NEXT CHARACTER
DONE:
                              ; DONE CODE
OVFL:
                              ; OVFL CODE
```

CO — Console Output Routine

The Console Output routine takes a single character and passes the character as a byte parameter if called from PL/M, or passed in the RC register if called from the assembler. The character is transmitted to the screen if ISIS-IV is operating in foreground, or to the file :CO: if ISIS-IV is executing in the background mode. RA and RC registers and CPU condition codes are affected.

1. PL/M CO Call Example

This example usesthe Console Output routine to output a string of characters. The routine terminates after a carriage return is detected in the output string and is transmitted to the console device. In this simple example there is no check to see if the buffer has been exhausted.

```
CO: PROCEDURE (CHAR) EXTERNAL;  /*ENTRY POINT INTO SYSTEM.LIB*/
 DECLARE CHAR BYTE;
 END CO;

DECLARE BUFFER(122) BYTE;       /*BUFFER CONTAINING STRING TO BE*/
                                /*OUTPUT*/
DECLARE INDEX BYTE;             /*INDEX INTO BUFFER*/
DECLARE CR LITERALLY '0DH';     /*CARRIAGE RETURN*/

INDEX = 0;
CALL CO(BUFFER(INDEX));         /*OUTPUT THE FIRST CHARACTER*/
DO WHILE BUFFER(INDEX) <> CR;
  INDEX = INDEX + 1;
  CALL CO(BUFFER(INDEX));       /*CONTINUE OUTPUTTING UNTIL A*/
                                /*CARRIAGE RETURN HAS BEEN OUTPUT*/
END;
```

2. Assembly Language CO Call Example

```
          EXTRN  CO        ; ENTRY POINT INTO SYSTEM.LIB
                           ; FOR CO
CR        EQU    0DH       ; CARRIAGE RETURN
BUFFER:   DS     122       ; BUFFER CONTAINING OUTPUT
                           ; STRING
;
          LXI    H,BUFFER  ; HL CONTAIN ADDRESS OF
                           ; BUFFER
LOOP:
          MOV    C,M       ; GET CHARACTER FROM
                           ; BUFFER
          CALL   CO        ; OUTPUT THE CHARACTER TO
                           ; THE CONSOLE
          MVI    A,CR
          CMP    M         ; IS IT A CARRIAGE RETURN?
          JZ     EXIT      ; GO TO EXIT IF IT IS
          INX    H         ; INCREMENT BUFFER POINTER
          JMP    LOOP      ; OUTPUT NEXT CHARACTER
EXIT:
```

IOCDR2 — Keyboard Interrupt Control

The K.I.C. routine transmits an address value in BC to K.I.C. The RB and RC registers and CPU condition codes are affected. The only commands accepted by this interface are those for enabling, disabling, and servicing keyboard interrupts that occur on level 6 of the Local Priority Interrupt Controller.

1. PL/M IOCDR2 Call Example

```
IOCDR2:
  PROCEDURE (VALUE) EXTERNAL;
    DECLARE VALUE ADDRESS
  END IOCDR2;

DECLARE KEYBD$ENABLE LITERALLY '0A02H';
DECLARE KEYBD$DISABLE LITERALLY '0A00H';
DECLARE KEYBD$SERVICE LITERALLY'0402H';

/* in main program*/
CALL IOCDR (Keyboard$Enable);

/* in interrupt service routine*/
CALL IOCDR (Keyboard$Service); /* clears interrupt request*/

/* in main program*/
CALL IOCDR2 (Keybd$disable);
```

2. Assembly Language IOCDR2 Example

```
;         IOCDR2  EQU       0F844H
          KBDEN   EQU       0A02H

                  LXI  B,   KBDEN
                  CALL      IOCDR2   ; enable keyboard intr.
```

System Status Routines

CSTS — Console Status Routine

The Console Input Status routine tests the ISIS-IV startup console to determine if a character is ready for input. If this routine is called from PL/M, it returns a value of 00 if no key has been pressed since the last call to the Console Input routine (CI), or a value of 0FF if a key has been pressed. If this routine is called from the assembler, then the 00 or 0FF value will be returned in the RA register.

1. **PL/M CSTS Call Example**

   The following example tests the ISIS-IV startup console during a Console Output operation so that the operator has the facility to signal that the output operation be terminated. A Control/C (03H) character entered at the console input device will signal this termination.

```
CSTS: PROCEDURE BYTE EXTERNAL;   /* ENTRY POINT INTO SYSTEM.LIB FOR*/
                                 /* CSTS */
 END CSTS;
CI: PROCEDURE BYTE EXTERNAL;     /* ENTRY POINT INTO SYSTEM.LIB */
                                 /* FOR CI*/
 END CI;

DECLARE CTLC LITERALLY '03H';    /* CONTROL/C SIGNALS TERMINATE */
                                 /* OPERATION */
IF CSTS THEN
   DO;                           /* A KEY HAS BEEN PRESSED */
    IF CI = CTLC THEN
       DO;
                                 /* CONTROL/C RECEIVED. TERMINATE */
                                 /* OUTPUT OPERATION. */

       END;
END;
```

2. **Assembly Language CSTS Call Example**

```
       EXTRN  CSTS   ; ENTRY POINT INTO SYSTEM.LIB FOR
                     CSTS
       EXTRN  CI     ; ENTRY POINT INTO SYSTEM.LIB FOR CI
CTLC   EQU    03H    ; CONTROL/C SIGNALS TERMINATE
                     ; OUTPUT

       CALL   CSTS   ; GET CONSOLE STATUS
       RRC           ; ROTATE TO CARRY FLAG
       JNC    CONT   ; NO CHARACTER, CONTINUE OUTPUT
                     ; OPERATION
       CALL   CI     ; THERE IS A CHARACTER, GET IT
       CPI    CTLC   ; IS IT A CONTROL/C
       JZ     TERM   ; IF YES, BRANCH TO TERMINATE CODE
CONT:
;
                     ; CODE TO CONTINUE OUTPUT
                     ; OPERATION


TERM:
                     ; CODE TO TERMINATE OUTPUT
                     ; OPERATION
```

MEMCK — Memory Check

The Check RAM Size routine returns the highest memory address of contiguous memory available to the user. This address is the highest address available after the Monitor has reserved its own memory (320 bytes) at the top of contiguous RAM. This value is returned as an address value (if called from PL/M) or in the RH and RL registers (if called from the assembler). MEMCK will always return the constant F6C0.

CAUTION

Remember that you can modify memory in the range 3180 to F6C0. The memory outside this range should not be changed or the monitor may malfunction and data and files may be destroyed.

1. PL/M MEMCK Call Example

The following example obtains the highest address of contiguous memory available.

```
MEMCK: PROCEDURE ADDRESS EXTERNAL;  /* ENTRY POINT INTO SYSTEM.LIB */
 END MEMCK;

DECLARE MADR ADDRESS;  /* ADDRESS TO CONTAIN VALUE RETURNED BY
                       /* MEMCK */

MADR = MEMCK;
```

2. Assembly Language MEMCK Call Example

```
        EXTRN  MEMCK  ; ENTRY POINT INTO SYSTEM.LIB FOR
                      ; MEMCK
MADR:   DS     2      ; CONTAINS VALUE RETURNED BY
                      ; MEMCK
;
        CALL   MEMCK
        SHLD   MADR   ; STORE ADDRESS IN MADR
```

CHAPTER 5 INTERRUPT PROCESSING

Interrupt processing is controlled by logic on the processor board. It provides an eight-level priority interrupt structure, using an Interrupt Mask Register (the I-register), and a current operating level indicator, which keeps track of the level of interrupt currently serviced. The Interrupt Mask Register is set by a program. You select which interrupts will be acknowledged at any time.

CAUTION

Do not write programs that use interrupts 0, 1, or 2 because the operating system uses these interrupts.

Interrupt Priorities

These are eight levels of interrupts, numbered 0 through 7. The levels correspond to the function keys 0 through 7 of Series IV. These function keys in the 8080/8085 execution mode correspond to the front panel interrupt switches of the MDS-800, the Series II and the Series III Development Systems.

Interrupt 0 has the highest priority and interrupt 7, the lowest. An interrupt is not serviced until all higher priority interrupts are serviced. An interrupt of level 4 that is currently being serviced can be interrupted to service an interrupt of level 3, 2, 1, or 0. It cannot be interrupted to service one of level 5, 6, or 7, nor can it be interrupted by another level 4.

Interrupt Mask Register

The Intellec Interrupt Mask Register (I-register) determines which interrupts are accepted by the system. The Interrupt Mask Register contains eight bits, each of which correspond to an interrupt level:

BITS	7	6	5	4	3	2	1	0
INTERRUPT LEVELS	7	6	5	4	3	2	1	0

A 1 bit in the Interrupt Mask Register prevents the corresponding interrupt from being serviced. A 0 bit allows the interrupt to be serviced. For example, the Intellec Monitor sets the Interrupt Mask Register to 0FEH (11111110B), which blocks all interrupts except interrupt 0.

The Interrupt Mask Register can be set programmatically by writing the desired value to Port 0FCH. For example,

```
MVIA,0F0H
OUT 0FCH
```

sets the Interrupt Mask Register to 11110000B, blocking interrupts 4 through 7 and allowing interrupts 0 through 3.

A program can also read the current value of the Interrupt Mask Register from Port 0FCH. For example,

```
IN 0FCH
```

places the current value of the Interrupt Mask Register into the A-register.

Interrupt Mask Register Initialization

The Interrupt Mask Register is initialized when ISIS-IV is loaded or re-initialized. The Mask Register is set to 0FCH (11111100B); interrupts 0 and 1 are allowed.

Interrupt Acceptance

When an interrupt occurs, the Interrupt Mask Register is checked to see if an interrupt of that level is permitted. If it is not, no further action is taken, but the interrupt is not cleared and remains pending. If the interrupt is permitted, the current operating level is checked to see if another interrupt of equal or higher priority is being serviced. If so, the new interrupt remains pending until the value of the current operating level is less than the priority of the new interrupt.

When the new interrupt can be serviced, all MULTIBUS® interrupts are locked out, while an RST instruction to the appropriate interrupt address (see the following list) is generated and the current operating level is set to the new value. The interrupt lockout is then removed.

The addresses called when an interrupt is accepted are

Interrupt Level	Address
0	0000H
1	0008H
2	0010H
3	0018H
4	0020H
5	0028H
6	0030H
7	0038H

Interrupt Removal

The program servicing an interrupt must do two things: transmit a signal to the interrupting device telling it to remove the interrupt signal it generated initially and send an end of interrupt signal to the 8259 programmable interrupt controller on the IEU (ISIS Execution Unit) board. If your program is using the interrupt (function) keys 3-7, it is not necessary to transmit a signal to the interrupting device because the circuitry does it automatically (auto-acknowledge). The end of the interrupt signal to the interrupt controller is sent by writing a value of 20H to port 0FDH *with the* *interrupts disabled.* (A stack overflow could result if the code permits another interrupt to be serviced while this is being done.) The following is a sample sequence in assembly language for doing this:

```
<remove interrupt signal from external device>
 DI                              ; DISABLE INTERRUPTS
 PUSH          PSLU              ; SAVE A-REGISTER AND FLAGS
 MVI           A,20H
 OUT           0FDH
 POP           PSW               ; RESTORE A-REGISTER AND FLAGS
 EI                              ; ENABLE INTERRUPTS
```

The following is a sample PL/M sequence for restoring the current operating level:

```
DISABLE;                         /*DISABLE INTERRUPTS*/
OUTPUT(0FDH)=20H;                /*RESTORE THE INTERRUPT LOGIC*/
ENABLE;                          /*ALLOW INTERRUPTS*/
```

The following example shows the skeleton of the code necessary to service an interrupt at level 5.

```
         ASEG                    ; VECTOR GOES AT ABSOLUTE
                                 ; LOCATION
         ORG           28H       ; RST ADDRESS FOR INTERRUPTS 5
         JMP           INTS
         CSEG                    ; PUT CODE IN RELOCATABLE CODE
                                 ; SEGMENT
INTS
         EI                      ; ROUTINE CAN BE INTERRUPTED
         PUSH          PSW       ; SAVE
         PUSH          B         ; REGISTERS
         PUSH          D         ;
         PUSH          H         ;
         .
         .
         .
<code to service interrupt and remove signal>

         .
         .
         .
         POP           H         ; RESTORE
         POP           D         ; REGISTERS
         POP           B         ;
         DI                      ; CRITICAL SECTION:
                                 ;               DISABLE INTERRUPTS
         MVI           A,20H     ; RESTORE CURRENT OPERATING LEVEL
         OUT           0FDH
         POP           PSW       ; RESTORE A REG AND FLAGS
         EI                      ; PERMIT INTERRUPTS AFTER NEXT
                                 ; INSTRUCTION
         RET                     ; THE RETURN MUST IMMEDIATELY
                                 ; FOLLOW THE EI TO MAKE SURE
                                 ; IT IS EXECUTED BEFORE ANOTHER
                                 ; INTERRUPT OCCURS
```

CHAPTER 6
ERROR CODES AND MESSAGES

This chapter describes the error codes and messages that are issued by ISIS-IV. See Appendix A for a complete listing.

Error Processing and Debugging

Errors encountered by ISIS-IV are displayed as decimal numbers and are either non-fatal or fatal. A non-fatal error occurs when you make a system call and an error code is returned. When a non-fatal error occurs, you may either call ERROR to display the error, or you may proceed to process the error.

The significance of each error number and identification of fatal errors are provided in Appendix A. If the error encountered is non-fatal, processing is immediately terminated and the error number is returned to the user program.

When ISIS-IV receives an error code from a lower level of the iNDX operating system, and ISIS can not process that error code, a fatal error will occur.

A fatal error will cause the following type of message to appear:

```
ERROR dd USER PC hhhh
```

where

dd	is the decimal number of the error.
hhhh	is the hexadecimal number of the return address in the calling program.

If the error number is 24, an additional message that displays the exception code message number, a device identifier, and the corresponding exception code message will appear, for example,

```
ERROR 24 USER PC hhhh
EXCEPTION 3201H: F11 ERR, ATTEMPT TO WRITE WITH
HARDWARE WRITE PROTECT SET
```

If the surface of the device is damaged, error 24 will occur, in which case you may wish to retain as much of the data as possible. This may be done by copying the files to a good device with the COPY command. See Chapter 2 for details on the COPY command and Appendix A for a complete list of exception code numbers and messages for error 24.

When you are executing ISIS-IV in the foreground, but not the import mode, the error will be sent to the screen. The error will be sent to the ISIS file :CO: when you are executing ISIS-IV in the background or the import mode.

The action taken in response to fatal errors depends on the setting of an internal system switch called the debug toggle. That switch indicates whether control is to return to ISIS-IV (debug=0) or to the Monitor (debug=1) when an error occurs.

If a fatal error occurs when you are executing ISIS-IV in the background or in the remote mode, the following will result:

- An error message is displayed.
- All files are closed.
- The 8085 resident section of ISIS-IV is reloaded.
- The ISIS Execution Unit (IEU) is re-initialized.
- All SUBMIT levels, including Series IV, are cancelled.
- ISIS Command Line Interpreter (CLI) will be initialized.

If a fatal error occurs when you are executing ISIS-IV in the foreground, but not the import mode, the following will result:

- ISIS-IV is exited.
- Command file execution is cancelled.

Either of the following actions sets the debug toggle to one and transfers control to the Monitor:

- Executing a DEBUG command
- Executing a LOAD system call with a transfer value of 2

Any of the following actions sets the debug toggle to zero, performs the operation listed, then transfers control to ISIS-IV:

- Pressing function key # 1 to generate an interrupt 1 while a program is running. This action terminates processing.
- Executing an EXIT system call. This action terminates a program.
- Executing a LOAD system call with a transfer value of 1. This action loads an absolute object file if ISIS-IV is still in memory.
- Executing a Monitor G8 command. This action exits the Monitor.

If the debug toggle is zero when a fatal error occurs, the following will occur:

- All open files are closed in their current state, including :CI: and :CO:.
- A fresh copy of ISIS-IV is read from the device, and ISIS-IV prompts for a command with a hyphen (-).

If the debug toggle is set when a fatal error occurs, the following will occur:

- All open files are left open.
- Control passes to the Monitor.
- The Monitor prompts for a command with a period (.).

At this point Monitor commands can be used to examine registers and memory to try to determine the cause of the error. However, the program cannot be restarted with any Monitor 85 G or N commands because the ISIS-IV restart address has not been saved. (A G or N command reloads ISIS.) *Do not reset the system at this point.*

NOTE

Although programs cannot be loaded in the ISIS-IV area, the ISIS-IV area is not protected from a running program. If a program should happen to destroy parts of ISIS-IV, subsequent system calls may not operate correctly and input/output may destroy areas on your device. This would happen mainly when an undebugged program is running.

ISIS-IV Error Message Codes

Error message numbers that are issued by ISIS-IV are allocated as follows:

- 1-99 inclusive—ISIS-IV resident routines (8080/8085 mode)
- 200-255 inclusive—non-resident system routines (8080/8085 mode)

Resident Routine Error Message Codes (8080/8085 Mode)

1. Fatal error. The memory area from 3100H to program origin is used for input/output buffers. Too few buffers were allocated to meet the current request in addition to earlier requests.
2. Illegal AFTN argument. The number supplied as an AFTN (active file table number) is inappropriate. Perhaps your program closed a file prematurely and then tried to read it.
3. Fatal error. AFT (active file table) is full. At most, six files may be active at one time. You must close one of your open files before a file can successfully be opened.
4. Incorrectly specified filename. You have possibly entered too many characters for filename, as in OLDFILE.1 (the maximum is six characters before the period, three after).
5. Unrecognized device name. You have entered an incorrect device name, as in :PR: for the line printer :LP:.
6. Attempt to write to input device. An attempt has been made to write to an input device. You can write only to an output device, such as a line printer (:LP:).
7. Fatal error. The disk is full. Check that you have specified the intended disk.
8. Attempt to read from output device. Some devices, like the line printer (:LP:), are output only and cannot be read. The current operation either should not be a READ or needs to use a different device name.
9. Directory is full. There is no room on the target directory to add an additional filename.
10. Pathname is not on same disk. A system call was attempted (RENAME) that requires two pathnames on the same device but the specified pathnames did not specify the same device.
11. File already exists. A filename identical to the one just used was found. Perhaps a different directory was intended, or a different spelling of the filename.
12. File is already open. Only console input (:CI:) and console output (:CO:) may be opened multiple times. If the spelling of the filename is correct, a flaw may exist in the program logic. For example, an earlier module may be using the file too soon or there may be an unintended loop.
13. No such file. The specified filename could not be found in the directory indicated by your command. A different directory may have the file.
14. Write-protected file encountered. The intended operation (e.g., WRITE, RENAME, DELETE) could not be done because the specified file has the write-protect attribute set.
15. Fatal error. ISIS overwrite. The system detected an attempt to write into the area reserved for the ISIS resident files, i.e., below 3100H. Such an operation would create unpredictable results and is disallowed.
16. Fatal error. Bad load format. This error was possibly caused by a source-language file. Files to be loaded for 8080/8085 execution must be in absolute object module format.
17. Not a disk file. An attempt was made to reference a disk file on a wrong device type with an improper pathname, such as :HP:FILE2 instead of :F*n*:FILE2.

18. Illegal ISIS commands. This error results when an ISIS system call is made with an illegal command number.
19. Attempted seek on non-disk file. Seeks on physical devices other than disk drives are invalid (:BB: is an exception and is valid).
20. Attempted back seek too far. The seek attempted to go beyond the beginning of the file; MARKER is set to zero.
21. Cannot rescan. The file was not opened for line-editing.
22. Illegal access mode to open. Only 1, 2, and 3 are valid, meaning input (read), output (write), or update (both read and write).
23. Missing filename. The system expected a filename, but one was not supplied.
24. Fatal error. Device input/output hardware error.
25. Illegal echo file. An echo file must have an active file table number (AFTN) between 0 and 255, and must already be opened for output. Check that these conditions are met.
26. Illegal attribute identifier. This error refers to the second parameter of the ATTRIB system call routine. Check that you have specified a valid parameter. Only 2 is valid, meaning the write-protect attribute.
27. Illegal seek command. An unsupported mode for the specified device was used in a seek command.
28. Missing extension. An expected file extension was not supplied.
29. Fatal error. Premature EOF. An unexpected end of file was encountered from the console.
30. Fatal error. Device specified was not ready.
31. Cannot seek on write only file. Seeks can be executed only on read or update files.
32. Cannot delete open file. You need to close the file before attempting to delete it. Verify the pathname.
33. Fatal error. Illegal system call parameter. A parameter was specified in a system call that is meant to be used as a pointer to a memory area intended for the receipt of data; however, ISIS found that this pointer was pointing to the memory space that ISIS occupies. ISIS will not allow a user to write into its memory space.
34. Fatal error. The return switch in a LOAD system call was not 0, 1, or 2, the only valid values.
35. Seek past EOF. An attempt was made to extend a file opened for input by seeking past end-of-file.
61. Device not assigned. The user has attempted to open a file on a logical device that has not been assigned to a directory. Use the ASSIGN command to map the device to the directory.
63. Synchronization error. Communication messages between ISIS and the Network Resource Manager are not synchronized. Reboot the ISIS software.
64. Network Comm error. The Network Resource Manager comm board is malfunctioning or missing.
65. Local Comm error. The Series IV comm board is malfunctioning or missing.
66. Illegal attribute for iNDX file. User has attempted to change the system, format, or invisible format attribute on an iNDX file. The Write attribute is the only attribute supported on iNDX files. The write-protect attribute is changed with the ATTRIB command or the ATTRIB system call.
70. iNDX file access right violation. User attempted to open an iNDX file without appropriate access rights to the file. File access rights can be changed with the ACCESS command.

71. Illegal operator on public file. User attempted to open for write access an iNDX public file that was being accessed by more than one user. When all other users have exited from the file, try again.
72. Maximum number of files on a device exceeded. User attempted to create more than the maximum number of files allowed for a particular device. The maximum number of files per device is determined when the System Generation procedure is run.
73. Attempt to delete a non-empty directory. User attempted to delete an iNDX directory that contains files. Delete all of the files in the directory before deleting the directory.
74. Illegal pathname syntax. User specified a fully qualified DFS pathname in a CREATE command and did not begin the pathname with a slash (/). All fully qualified pathnames must begin with a slash (/).
75. Non-terminating path element is not a directory. User specified a fully qualified pathname in a CREATE command where one of the first elements of the pathname was a data file. Only the final element of the pathname can be a data file.
76. Attempt to create a connected iNDX file. User assigned :F*x*: to /ROOT/A and then attempted to CREATE /ROOT/A. No two directories can have the same pathname.
77. Username/Password mismatch. User attempted to LOGON with a username that is valid but an incorrect password for this user.
78. Username not known. User attempted to LOGON with an unknown username.
79. File error on system file. The system accessed the user definition file. This file is a system file only.
80. iNDX file detached, device dismounted. While a user was accessing a file, the device on which the file resided was dismounted or removed at the Network Resource Manager.
81. Maximum remote attaches exceeded. User attempted to exceed the maximum number of files that could be attached at one time. The user can attach only 12 remote files at once.
82. Illegal password syntax. Attempt to LOGON with a password of more than 14 characters.
83. Illegal username syntax. Attempt to LOGON with a username of more than 14 characters.
88. Remote error that does not map into ISIS error numbers.

Non-Resident Routine Error Message Codes (8080/8085 Mode)

201. Unrecognized switch. Certain predefined switches (e.g., E, O, Q, and B) can be used depending on the ISIS-IV command. Some commands that have switches are DIR and COPY.
202. Unrecognized delimiter. A character was encountered that was invalid in a name and not known as a delimiter.
203. Invalid syntax. There is an error in the command as entered. The error may be an unrecognized keyword or a missing comma.
206. Illegal disk label. The label supplied violates the rules for a valid disk label.
208. Checksum error. The bits of the records read do not add up properly. An inappropriate input or medium was supplied. There may be an error in the internal format of the specified file that may have occurred during translation or linking. Retranslate and relink the source module.

209. Relocation file sequence error. An inappropriate input file was specified.
210. Insufficient memory. The required amount of RAM is not present.
211. Record too long. A record longer than allowed was encountered.
212. Illegal relocation type. Relocation types must conform to Intel standard formats.
213. Fixup bounds error. The required address violated numeric bounds on addresses.
214. Illegal SUBMIT parameter. An error was made in the actual parameter to be substituted for a formal parameter in a command sequence file.
215. Argument too long. The number of characters in the actual argument must not exceed 31.
216. Too many parameters. More parameters were supplied than defined.
217. Object record too short. This error may be caused by an I/O error in the file to be loaded.
218. Illegal record format. The record format did not match the Intel standard.
219. Phase error. The expected phase input (e.g., for the next step of a translation process) was not correctly supplied.
220. No end-of-file record in object module file. There is an error in the internal format of the specified file. Retranslate and relink the source module.
221. Segment overflow during LINK operation. The output segment cannot be greater than 64K bytes.
222. Unrecognized record in object module file. There is an error in the internal format of the specified file. Retranslate and relink the source module.
223. Fixup record pointer is incorrect. There is an error in the internal format of the specified file. Retranslate and relink the source module.
224. Illegal record sequence in object module file in LINK. There is an error in the internal format of the specified file that may have occurred during translation. Retranslate and relink the source module.
225. Illegal module name specified. An illegal or misspelled module name was entered.
226. Module name exeeds 31 characters. Module names exeeding 31 characters may not be used.
227. Command syntax requires left parenthesis. There is a missing left parenthesis in the command line. Re-enter the command correctly.
228. Command syntax requires right parenthesis. There is a missing right parenthesis in the command line. Re-enter the command correctly.
229. Unrecognized control specified in command. A character string other than the expected control keyword was entered. Enter the correct control keyword.
230. Duplicate symbol found. You have attempted to add a symbol that already exists.
231. File already exists. The file specified in a CREATE command already exists.
232. Unrecognized command. An illegal or misspelled command was entered.
233. Command syntax requires a TO clause. The command syntax requires a TO clause to specify the output file.
234. Filename illegally duplicated in command. The same filename is specified both as an input and output file.
235. File specified in command is not a library file. The specified file is not a library file.
236. More than 249 common segments in input files. You cannot have more than 249 common segments.
237. Specified common segment not found in object file. The input module does not contain the common segment specified in the command.

238. Illegal stack content record in object file. There is an error in the internal format of the specified file that may have occurred during the translation and link process. Retranslate and relink the source module.

239. No module header in input object file. There is an error in the internal format of the specified file. Retranslate and relink the source module.

240. Program exeeds 64K bytes. The output module to be placed in the output file exeeds the maximum of 64K bytes.

Additional 8080/8085 link and locate error messages are described in the *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems.*

APPENDIX A
SUMMARY OF ERROR MESSAGES

This appendix provides a list of error codes and messages issued by ISIS-IV, and some non-resident system routines. For further information on error codes refer to the following manuals:

Error Codes	Manuals
101-149	*DEBUG-88 User's Guide*, 121758
201-240	*MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems*, 121617

Resident Routine Error Message Codes (8080/8085 Mode)

1. Fatal error. Too few buffers were allocated.
2. Illegal active file table number.
3. Fatal error. Active file table is full.
4. Incorrectly specified filename.
5. Unrecognized device name.
6. Attempt to write to input device.
7. Fatal error. The device is full.
8. Attempt to read from output device.
9. Directory is full.
10. Pathname is not on same device.
11. File already exists.
12. File is already open.
13. No such file.
14. Write-protected file encountered.
15. Fatal error. ISIS overwrite.
16. Fatal error. Bad load format.
17. Not a device file.
18. Illegal ISIS commands.
19. Attempted seek on non-disk file.
20. Attempted back seek too far.
21. Cannot rescan.
22. Illegal access mode to open.
23. Missing filename.
24. Fatal error. Device input/output hardware error.
25. Illegal echo file.
26. Illegal attribute identifier.
27. Illegal seek command.
28. Missing extension.

29. Fatal error. Premature EOF.
30. Fatal error. Device not ready.
31. Cannot seek on write only file.
32. Cannot delete open file.
33. Fatal error. Illegal system call parameter.
34. Fatal error. Invalid return switch in a LOAD system call.
35. Seek past EOF.
61. Device not assigned.
62. Reserved.
63. Synchronization error.
64. Network Comm error.
65. Local Comm error.
66. Illegal attribute for iNDX file.
70. iNDX file access right violation.
71. Illegal operation on public file.
72. Maximum number of files on a device exceeded.
73. Attempt to delete a non-empty directory.
74. Illegal pathname syntax.
75. Non-terminating path element is not a directory.
76. Attempt to create a connected iNDX file.
77. Username/Password mismatch.
78. Username not known.
79. File error on system file.
80. iNDX file detached, device dismounted.
81. Maximum remote attaches exceeded.
82. Illegal password syntax.
83. Illegal username syntax.
88. Unknown remote error.

NOTE

When error 24 occurs, an additional message that reports the hexadecimal exception code number of the operating system, the device identifier, and the corresponding error message will appear.

```
EXCEPTION xxxxH: yyyERR, zzz
```

where

*xxxx*H	is the hexadecimal number of the exception code of the operating system. When the last digit of the exception code number is a 0 or a 1 (i.e., 3011H), the error message will be for a miniature flexible diskette. If the last digit is a 2, 3, 4, or 5 (i.e., 3013H), the error will be for a hard disk. If the last digit is a 6 (i.e., 3016H), the error message will be for a Winchester device.

yyy	is the device identifier (i.e., Fl1 identifies a miniature flexible diskette).
zzz	is the corresponding error message (i.e, ATTEMPT TO WRITE WITH HARDWARE WRITE PROTECT SET) that has the following meaning:

For a miniature flexible diskette if *n*=0 or 1:

```
301n   E$B$DELETED$DAM
302n   E$B$DATA$CRC
304n   E$B$ID$CYL$MIS
308n   E$B$SECTOR$BOUNDS
30An   E$B$ID$CRC
30En   E$B$NO$ID$AM
30Fn   E$B$BAD$DATA$AM
310n   E$B$DATA$OVERRUN
320n   E$B$WRITE$PROTECT
340n   E$B$DRIVE$FAULT
370n   E$B$NO$SECTOR
371n   E$B$BAD$TRACK
372n   E$B$SEEK$MIS
378n   E$B$UNEXPECTED
380n   E$B$DEVICE$NOT$READY
```

For a hard disk if *n*=2, 3, 4, or 5:

```
301n   ID field miscompare
302n   Data Field CRC
304n   Seek error
308n   Sector address out of bounds
30An   ID field CRC
30Bn   Protocol error
30Cn   Cyl addr out of bounds
30En   Sector not found
30Fn   No data field data mark
310n   Format overrun
320n   Write protected
340n   Drive write error
380n   Drive not ready
```

For a 35 megabyte Winchester if *n*=6:

```
302n   Data field ECC
304n   Drive Seek error
30An   ID field ECC
30En   Sector not found
311n   Controller RAM fail
312n   Controller ROM fail
313n   Seek in progress
314n   Track type disallows operation
315n   Beyond end of media
316n   Illegal sector size
317n   Controller diagnostic fault
318n   No index signal
319n   Invalid function code
31An   Invalid address
320n   Write protected
340n   Drive fault
372n   ID Cylinder address does not match seek
```

378*n*	Unexpected error code
380*n*	Drive not ready

3C00	E$B$BAD$COMMAND$CODE
3C10	E$B$BLOCK$OVERFLOW
3C20	E$B$72$INVALID
3C30	E$B$MINI$Q
3C40	EB8089
3C50	E$B$NO$CONTROLLER
3CF0	E$B$72$PROTOCOL

3D80	E$KEYBOARD$ABORT
3DC0	E$B$5440$PROTOCOL
3DC1	E$B$5440$Q
3DC2	E$B$5440$CONFIG
3DC3	E$B$UNKNOWN$INT
3DC4	E$B46UF$SEG$OVERFLOW
3DE0	E$B$SBIOS$ACCESS
3DF0	E$INSUFFICIENT$MIP$HEADERS
3E01	E$B$PRINTER$TIMEOUT
3E02	E$B$PRINTER$FAULT
3E04	E$B$PRINTER$NOT$READY
3E10	E$B$PRINTER$PROTOCOL
3F00	E$B$WINC$DATA$STRUC
3F01	E$B$WINC$PROTOCOL

4000	E$OPEN
4001	E$NOPEN
4002	E$FTYPE
4003	E$SYNTAX
4004	E$DEVICE$NOT$READY
4005	E$DEVICE$IO$ERROR
4006	E$COMM$ERROR
4007	E$NODE$NOT$READY
4008	E$MARKED$DELETED
4009	E$OPEN$MODE

4010	E$CONNECTIONS$EXIST
4011	E$DIR$NOT$EMPTY
4012	E$LIMIT
4013	E$EXIST
4014	E$CONSOLE
4015	E$LOG$NAME$EXISTS
4016	E$ILLEGAL$DEVICE$ID
4017	E$SYSTEM$DEVICE
4018	E$LOG$NAME$DOES$NOT$EXIST
4019	E$BAD$PATH

401A	E$NOT$DIRECTORY
401B	E$PATH$DOES$NOT$EXIST
401C	E$ATTACHED
401D	E$DETACHED
401E	E$NETWORK

4FFF	E$CONSISTENCY

8004	E$PARAM
CCB6	M$DISALLOWED$QUERY
CCB7	M$SINGLE$COMP
CCB8	M$DIR$EXIST
CCB9	M$DEL$CREATE$CONFLICT
CCBA	M$DELETE$ACCESS
CCBB	M$DIR$REQD
CCBC	M$DISPLAY$ACCESS
CCBD	M$ADD$ACCESS
CCBE	M$ILLEGAL$WILDCARD
CCBF	M$DISALLOWED$WILDCARD
CCC0	M$PARENT$NEXIST
CCC1	M$DATA$DIR$OPTIONS
CCC2	M$DIR$OPTION
CCC3	M$DATA$OPTION
CCC4	M$DIR$CREATE
CCC5	M$PASSWORD
CCC6	M$ILLEGAL$VALUE
CCC7	M$USER$ID
CCC8	M$USER$NAME
CCC9	M$NOT$COMPLETED
CCCA	M$CANNOT$CREATE
CCCB	M$READ$ACCESS
CCCC	M$COMMAND$SYNTAX
CCCD	M$SYSTEM$ERROR
CCCE	M$PATH$SYNTAX
DFFD	E$ISIS$WP$CONSISTENCY
DFFE	E$VNEXIST
DFFF	E$USER$SUPPORT
E000	E$SYN$SCAN$OVF
E001	E$SYN$PARSE$OVF
E002	E$SYN$REMOVE$OVF
E003	E$SYN$BUF$OVF
E004	E$SYN$VERSION
E005	E$SYN$CONSISTENCY
E11A	Bad parameter to monitor routine
E119	Unsupported monitor function was called
E11D	No IEU memory
E115	Cannot load ISIS.LM
E111	IEU hardware not responding
E10B	IEU board not responding (damaged software possible)
FFEB	E$PASSWORD$MISMATCH
FFEC	E$USER$UNKNOWN
FFED	E$USER$ID
FFEE	E$SUPER$USER
FFEF	E$FATAL$UDF$FNEXIST
FFF0	E$FATAL$IO$ERROR
FFF1	E$REBOOT$REQUIRED
FFF2	E$MULTIPLE$ID
FFF3	E$MULTIPLE$NAME
FFF4	E$SYSTEM$FILE
FFF5	E$IO
FFF9	E$BOUNDS
FFFA	E$FILE$TOO$LONG
FFFB	E$DELETED
FFFC	E$END$OF$FILE

Non-Resident Routine Error Message Codes (8080/8085 Mode)

201. Unrecognized switch.
202. Unrecognized delimiter.
203. Invalid syntax.
206. Illegal disk label.
208. Checksum error.
209. Relocation file sequence error.
210. Insufficient memory.
211. Record too long.
212. Illegal relocation type.
213. Fixup bounds error.
214. Illegal SUBMIT parameter.
215. Argument too long.
216. Too many parameters.
217. Object record too short.
218. Illegal record format.
219. Phase error.
220. No EOF record in object module file.
221. Segment overflow during LINK operation.
222. Unrecognized record in object module file.
223. Fixup record pointer is incorrect.
224. Illegal record sequence in object module file in LINK.
225. Illegal module name specified.
226. Module name exceeds 31 characters.
227. Command syntax requires left parenthesis.
228. Command syntax requires right parenthesis.
229. Unrecognized control specified in command.
230. Duplicate symbol found.
231. File already exists.
232. Unrecognized command.
233. Command syntax requires a TO clause.
234. Filename illegally duplicated in command.
235. File specified in command is not a library file.
236. More than 249 common segments in input files.
237. Specified common segment not found in object file.
238. Illegal stack content record in object file.
239. No module header in input object file.
240. Program exceeds 64K bytes.

Table A-1. Non-Fatal Error Numbers Returned by System Calls

System Call	Error Number
OPEN	3, 4, 5, 9, 12, 13, 14, 22, 23, 25, 28, 52, 54, 61, 70
READ	2, 8
WRITE	2, 6
SEEK	2, 19, 20, 27, 31, 35
RESCAN	2, 21
CLOSE	2
DELETE	4, 5, 13, 14, 17, 23, 28, 32, 61, 70
RENAME	4, 5, 10, 11, 13, 17, 23, 28, 52, 54, 56, 61, 70
ATTRIB	4, 5, 13, 23, 26, 28, 52, 54, 61, 66, 70
CONSOL	None; all errors are fatal
WHOCON	None
ERROR	None
LOAD	3, 4, 5, 12, 13, 22, 23, 28, 34, 61, 70
EXIT	None
SPATH	4, 5, 23, 28, 61, 70
GETD	3, 4, 5, 13, 23
GETATT	4, 5, 13, 23, 28

Table A-2. Fatal Errors Issued by System Calls

System Call	Error Number
OPEN	1, 7, 24, 30, 33, 51, 63, 64, 65
READ	24, 30, 33, 63, 64, 65
WRITE	7, 24, 30, 33, 63, 64, 65
SEEK	7, 24, 30, 33, 63, 64, 65
RESCAN	33, 63, 64, 65
CLOSE	33, 63, 64, 65
DELETE	1, 24, 30, 33, 63, 64, 65
RENAME	1, 24, 30, 33, 51, 63, 64, 65
ATTRIB	1, 24, 30, 33, 51, 63, 64, 65
CONSOL	1, 4, 5, 12, 13, 14, 22, 23, 24, 28, 30, 33
WHOCON	33
ERROR	33
LOAD	1, 15, 16, 24, 30, 33, 63, 64, 65
SPATH	33
GETD	1, 24, 30, 33
GETATT	1, 24, 30, 33

APPENDIX B
COMMAND SYNTAX SUMMARY

This appendix lists the ISIS-IV commands and syntax alphabetically. Chapter 2 of this manual contains complete descriptions and examples of each command.

ACCESS

```
        { :Fn:      }
ACCESS  { pathname/ }  filename [switch]<cr>
```

ASSIGN

```
                   [    { fully qualified directory name, } ]
ASSIGN  [:Fn:]     [ TO { NULL                            } ]<cr>
```

ATTRIB

```
ATTRIB  [:Fn:]filename[attriblist]  [Q]<cr>
```

COPY

```
                              { [:Fn:][outfile] }
COPY  [Fn:]infile [,...]  TO  { :device:        }  [switches]<cr>
```

CREATE

```
        { :Fn:      }
CREATE  { pathname/ }  new directory name<cr>
```

DELETE

```
DELETE  [:Fn:] filename[Q]  [,...[Q]]<cr>
```

DIR

```
DIR  [FOR  filename]  [TO  listfile]  [switches]<cr>
```

EXIT

```
EXIT<cr>
```

REMOVE

```
REMOVE { :Fn:      } directory name<cr>
       { pathname/ }
```

RENAME

```
RENAME [:Fn:]oldname TO [:Fn:]newname<cr>
```

SPACE

```
SPACE /volume name<cr>
```

SUBMIT

```
SUBMIT [:Fn]filename [(parameter[,...])]<cr>
```

VERS

```
VERS command<cr>
```

WHO

```
WHO<cr>
```

APPENDIX C MONITOR COMMAND SUMMARY

This appendix provides a summary of the command syntax for the ISIS Execution Unit Monitor (MON 85) Commands.

C — Compare Command

C *range*, *destination-address*< c r >

D — Display Command

[count]D $\left[\begin{Bmatrix}\mathtt{W}\\ \mathtt{X}\end{Bmatrix}\right]$ [*range*][,]< c r >

F — Find Command

F *range* , *data*< c r >

G — Execute Command

G[start-address] [, {*break-address*}]< c r >

M — Move Command

M *range*, *destination-address*< c r >

N — Single Step Command

[count]N [P][*start-address*][,]< c r >

P — Print Value Command

P $\left[\begin{Bmatrix}\mathtt{T}\\ \mathtt{S}\end{Bmatrix}\right]$ [{*address*| *expression*| *literal*}][,] . . . < c r >

S — Substitute Command

[count] S [W] *start-address*[= *expression*| *expression*]...[,]< c r >

X — Examine Register Command

X[*register*[= *expression*]]< c r >

APPENDIX D
ASCII CODES

Table D-1. ASCII Code List

Decimal	Octal	Hexadecimal	Character
0	000	00	NUL
1	001	01	SOH
2	002	02	STX
3	003	03	ETX
4	004	04	EOT
5	005	05	ENQ
6	006	06	ACK
7	007	07	BEL
8	010	08	BS
9	011	09	HT
10	012	0A	LF
11	013	0B	VT
12	014	0C	FF
13	015	0D	CR
14	016	0E	SO
15	017	0F	SI
16	020	10	DLE
17	021	11	DC1
18	022	12	DC2
19	023	13	DC3
20	024	14	DC4
21	025	15	NAK
22	026	16	SYN
23	027	17	ETB
24	030	18	CAN
25	031	19	EM
26	032	1A	SUB
27	033	1B	ESC
28	034	1C	FS
29	035	1D	GS
30	036	1E	RS
31	037	1F	US
32	040	20	SP
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	—
46	056	2E	.
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<

Table D-1. ASCII Code List (Cont'd.)

Decimal	Octal	Hexadecimal	Character
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	∧
95	137	5F	—
96	140	60	'
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~
127	177	7F	DEL

Table D-2. ASCII Code Definition

Abbreviation	Meaning	Decimal Code
NUL	NULL Character	0
SOH	Start of Heading	1
STX	Start of Text	2
ETX	End of Text	3
EOT	End of Transmission	4
ENQ	Enquiry	5
ACK	Acknowledge	6
BEL	Bell	7
BS	Backspace	8
HT	Horizontal Tabulation	9
LF	Line Feed	10
VT	Vertical Tabulation	11
FF	Form Feed	12
CR	Carriage Return	13
SO	Shift Out	14
SI	Shift In	15
DLE	Data Link Escape	16
DC1	Device Control 1	17
DC2	Device Control 2	18
DC3	Device Control 3	19
DC4	Device Control 4	20
NAK	Negative Acknowledge	21
SYN	Synchronous Idle	22
ETB	End of Transmission Block	23
CAN	Cancel	24
EM	End of Medium	25
SUB	Substitute	26
ESC	Escape	27
FS	File Separator	28
GS	Group Separator	29
RS	Record Separator	30
US	Unit Separator	31
SP	Space	32
DEL	Delete	127

APPENDIX E
HEXADECIMAL-DECIMAL CONVERSION

The following table is for hexadecimal to decimal and decimal to hexadecimal conversion. To find the decimal equivalent of a hexadecimal number, locate the hexadecimal number in the correct position and note the decimal equivalent. Add the decimal numbers.

To find the hexadecimal equivalent of a decimal number, locate the next lower decimal number in table E-1 and note the hexadecimal number and its position. Subtract the decimal number from the table from the starting number. Find the difference in the table. Continue this process until there is no difference.

Table E-1. Hexadecimal-Decimal Conversion

Most Significant Byte				Least Significant Byte			
Digit 4		Digit 3		Digit 2		Digit 1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4 096	1	256	1	16	1	1
2	8 192	2	512	2	32	2	2
3	12 288	3	768	3	48	3	3
4	16 384	4	1 024	4	64	4	4
5	20 480	5	1 280	5	80	5	5
6	24 576	6	1 536	6	96	6	6
7	28 672	7	1 792	7	112	7	7
8	32 768	8	2 048	8	128	8	8
9	36 864	9	2 304	9	144	9	9
A	40 960	A	2 560	A	160	A	10
B	45 056	B	2 816	B	176	B	11
C	49 152	C	3 072	C	192	C	12
D	53 248	D	3 328	D	208	D	13
E	57 344	E	3 548	E	224	E	14
F	61 440	F	3 840	F	240	F	15

INDEX

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating).________

NAME ________ DATE ________
TITLE ________
COMPANY NAME/DEPARTMENT ________
ADDRESS ________
CITY ________ STATE ________ ZIP CODE ________
(COUNTRY)

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS ...

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

Intel Corporation
Attn: Technical Publications M/S 6-2000
3065 Bowers Avenue
Santa Clara, CA 95051

INTEL CORPORATION, 3065 Bowes Avenue, Santa Clara, California 95051 (408) **987-8080**

Printed in U.S.A.