

**GETTING STARTED
WITH THE iRMX™86
SYSTEM**

Order Number: 144349-001

REV.	REVISION HISTORY	PRINT DATE
-001	Original Issue	3/82

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The Information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined as ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BXP	Insite	iSBC	Multibus
CREDIT	Intel	iSBX	Multimodule
i	intel	Library Manager	Plug-A-Bubble
ICE	Intelevison	MCS	PROMPT
iCS	Intellec	Megachassis	RMX/80
im	iOSP	Micromainframe	System 2000
iMMX	iRMX	Micromap	UPI

PREFACE

Here is the information you will need in order to use the Configured iRMX 86 Operating System (iRMX 86 PC), a ready-to-use version of Intel's general, configurable iRMX 86 Operating System. "Ready-to-use" means that Intel has selected the individual software features and then put this system software together. In order to do so, we have made some assumptions about the hardware on which you will run the system, and we describe this hardware in the manual. The iRMX 86 PC product allows you to use the Operating System as soon as you have the correct hardware environment, without going through the configuration process.

READERS AND CONTENTS OF THIS MANUAL

Except for Chapter 5, this manual is written for programmers who will use the Operating System. Chapter 5 is written for a hardware technician or engineer who assembles the hardware, if the programmer is not the person who does this.

Here is how the manual is organized.

- Chapter 1 OVERVIEW. This chapter describes the characteristics of the Operating System. You will want to read this chapter to become familiar with iRMX 86 concepts and terms.
- Chapter 2 USING THE SYSTEM. This chapter shows how to start up (bootstrap load) the Operating System, and shows examples of iRMX 86 Commands.
- Chapter 3 iRMX 86 COMMANDS. This chapter contains complete descriptions of the iRMX 86 Commands, arranged alphabetically.
- Chapter 4 UDI SYSTEM CALLS. This chapter contains general information about the Universal Development Interface (UDI), followed by descriptions of each UDI System Call.
- Chapter 5 PREPARING YOUR HARDWARE. This chapter describes the hardware required to run the Operating System, and how to prepare that hardware.

PREFACE (continued)

- Appendix A This appendix lists the codes that the iRMX 86 Operating System uses to indicate exceptional conditions, such as hardware failures and mistakes in how a program uses the system.
- Appendix B This appendix provides a list of "internal" iRMX 86 System Calls. You will not need to use these system calls to write and run programs. But the information in this appendix provides an overview of the services provided by the iRMX 86 Operating System.
- Appendix C This appendix describes how to use the monitor that is delivered as part of the iRMX 86 PC System.

WHAT YOU GET

The iRMX 86 PC Release Package contains:

- The System Diskette, which is labeled
Preconfigured iRMX 86 Operating System
- The Library Diskette, which is labeled
iRMX 86 Interface Libraries
- Four (4) EPROM devices which contain the Monitor and a Bootstrap Loader (you install these on your iAPX 86 Single Board Computer)
- This manual
- One Software Problem Report Form
- One Software Registration Card, which you should complete and return when you receive the package

PREFACE (continued)

RELATED PUBLICATIONS

The following manuals provide additional information that may be helpful to users of this manual. These manuals are described in Chapter 6.

<u>Manual</u>	<u>Number</u>
Introduction to the iRMX™ 86 Operating System	9803124
iRMX™ 86 Nucleus Reference Manual	9803122
iRMX™ 86 Basic I/O System Reference Manual	9803123
iRMX™ 86 Extended I/O System Reference Manual	143308
iRMX™ 86 Loader Reference Manual	143318
iRMX™ 86 Human Interface Reference Manual	9803202
iRMX™ 86 Disk Verification Utility Reference Manual	144133
iRMX™ 86 System Programmer's Reference Manual	142721
iRMX™ 86 Programming Techniques Manual	142982
Guide to Writing Device Drivers for the iRMX™ 86 and iRMX™ 88 I/O Systems	142926
iRMX™ 86 Configuration Guide	9803126
iRMX™ 86 Installation Guide	9803125
EDIT Reference Manual	143587
Guide to Using iRMX™ 86 Languages	143907
8086/8087/8088 Macro Assembly Language Reference Manual for 8086-Based Development Systems	121627
8086/8087/8088 Macro Assembler Operating Instructions for 8086-Based Development Systems	121628
PL/M-86 User's Guide for 8086-Based Development Systems	121636

PREFACE (continued)

<u>Manual</u>	<u>Number</u>
FORTRAN-86 User's Guide	121570
Pascal-86 User's Guide	121539
iAPX 86,88 Family Utilities User's Guide	121616
Run-Time Support Manual for iAPX 86, 88 Applications	121776
User's Guide for the iSBC® 957B iAPX 86, 88 Interface and Execution Package	143979
iSBC® 86/12A Single Board Computer Hardware Reference Manual	9803074
iSBC® 86/14 and iSBC® 86/30 Single Board Computer Hardware Reference Manual	144044
iSBX™ 208 Flexible Disk Drive Controller Hardware Reference Manual	143078
iSBC® 337 Multimodule™ Numeric Data Processor Hardware Reference Manual	142887

CONTENTS

	PAGE
CHAPTER 1	
SYSTEM OVERVIEW	
Hardware Environment For the iRMX 86 PC System.....	1-2
Language Translators and Utilities.....	1-4
Universal Development Interface.....	1-5
The iRMX 86 File System.....	1-5
Hierarchical Naming of Files.....	1-5
iRMX 86 File Terminology.....	1-6
Device Logical Names.....	1-7
File Operations From a Terminal.....	1-8
File Operations From Programs.....	1-8
Files and Directories with the iRMX 86 PC Operating System.....	1-9
System Diskette.....	1-9
Default Directory (\$).....	1-10
System Directory (SYSTEM).....	1-10
Program Directory (PROG).....	1-10
Work Directory (WORK).....	1-11
Library Diskette.....	1-11
Program Loading.....	1-11
Bootstrap Loading.....	1-11
Monitor.....	1-12
Selective Error Processing.....	1-12
Summary.....	1-12
CHAPTER 2	
USING THE SYSTEM	
Starting the System.....	2-2
Invoking iRMX 86 Commands.....	2-3
Preposition Parameters.....	2-5
Terminal Controls.....	2-6
Unequal Number of Files in Input and Output Lists.....	2-7
More Input Files Than Output Files.....	2-7
More Output Files Than Input Files.....	2-8
Safeguards.....	2-8
Example Commands.....	2-8
How To Set the System Date and Time.....	2-8
How to Display the Contents of a Directory.....	2-10
Using the DIR Command with No Parameters.....	2-10
Directory Displayed in an Alternate Format.....	2-12
Directory Listing of System Directory.....	2-13
Directory Listing of SYSTEM/UDI.....	2-14
How To Copy Files.....	2-14
Creating a New Copy of a File.....	2-15
Copying Multiple Files With One Command.....	2-15
Copying One File OVER Another.....	2-16
Creating a Directory.....	2-17
Displaying The Contents of a File On The Terminal.....	2-17
Giving a File a New Name.....	2-18
How to Make Copies of Your System Diskette.....	2-19

CONTENTS (continued)

	PAGE
CHAPTER 3	
iRMX 86 COMMANDS	
Command Syntax Schematics.....	3-1
Command Dictionary.....	3-3
ATTACHDEVICE.....	3-5
BACKUP.....	3-8
COPY.....	3-15
CREATEDIR.....	3-19
DATE.....	3-20
DEBUG.....	3-21
DELETE.....	3-22
DETACHDEVICE.....	3-24
DIR.....	3-25
DISKVERIFY.....	3-32
DOWNCOPY.....	3-37
FORMAT.....	3-40
RENAME.....	3-45
RESTORE.....	3-48
SUBMIT.....	3-55
TIME.....	3-58
UPCOPY.....	3-59
CHAPTER 4	
UDI SYSTEM CALLS	
Using the UDI.....	4-2
UDI Libraries.....	4-2
Include Files.....	4-2
Exceptional Conditions.....	4-2
Data Types.....	4-3
Descriptions of System Calls.....	4-4
Memory Management System Calls.....	4-4
File-Handling System Calls.....	4-5
Exception-Handling System Calls.....	4-6
System Call Dictionary.....	4-7
DQ\$ALLOCATE.....	4-9
DQ\$ATTACH.....	4-11
DQ\$CHANGE\$EXTENSION.....	4-12
DQ\$CLOSE.....	4-13
DQ\$CREATE.....	4-14
DQ\$DECODE\$EXCEPTION.....	4-15
DQ\$DELETE.....	4-16
DQ\$DETACH.....	4-17
DQ\$EXIT.....	4-18
DQ\$FREE.....	4-19
DQ\$GET\$ARGUMENT.....	4-20
DQ\$GET\$CONNECTION\$STATUS.....	4-22
DQ\$GET\$EXCEPTION\$HANDLER.....	4-24
DQ\$GET\$SIZE.....	4-25
DQ\$GET\$SYSTEM\$ID.....	4-26
DQ\$GET\$TIME.....	4-27

CONTENTS (continued)

	PAGE
CHAPTER 4 (continued)	
DQ\$OPEN.....	4-28
DQ\$OVERLAY.....	4-30
DQ\$READ.....	4-32
DQ\$RENAME.....	4-34
DQ\$SEEK.....	4-35
DQ\$SPECIAL.....	4-37
DQ\$SWITCH\$BUFFER.....	4-39
DQ\$TRAP\$EXCEPTION.....	4-40
DQ\$TRUNCATE.....	4-41
DQ\$WRITE.....	4-42
Example Program.....	4-44
CHAPTER 5	
PREPARING YOUR HARDWARE	
The iRMX 86 PC Hardware Environment.....	5-2
Single Board Computer.....	5-2
Flexible Diskette Controller And Drives.....	5-3
Memory.....	5-3
Line Printer.....	5-3
iSBC 957B Package.....	5-5
Modifying Boards.....	5-5
Modifying the iSBC 208 Controller.....	5-6
Modifying the iSBC 86/12A Single Board Computer.....	5-7
Interrupt Level Jumpers.....	5-7
Additional Jumpers.....	5-7
Parallel Port.....	5-8
Switch Settings.....	5-8
Devices.....	5-9
Modifying the iSBC 86/14 Single Board Computer.....	5-10
Jumpers.....	5-10
Devices.....	5-11
Modifying the iSBC 86/30 Single Board Computer.....	5-12
Jumpers.....	5-12
Devices.....	5-13
Convenience Charts.....	5-14
CHAPTER 6	
DOCUMENTATION	
This Manual.....	6-1
iRMX 86 Manuals.....	6-1
Language Translators and Utilities Manuals.....	6-4
Hardware Manuals.....	6-6
APPENDIX A	
iRMX 86 EXCEPTION CODES.....	A-1

CONTENTS (continued)

	PAGE
APPENDIX B	
iRMX 86 SYSTEM CALLS	
Layers of the iRMX 86 System.....	B-1
Nucleus System Calls.....	B-3
Basic I/O System Calls.....	B-5
Extended I/O System Calls.....	B-6
Human Interface System Calls.....	B-7
System Programmer System Calls.....	B-8

APPENDIX C

MONITOR COMMANDS

Command Structure.....	C-2
Byte and Word Variables.....	C-2
Numeric (Real, Integer and BCD) Variables.....	C-3
Address Specification.....	C-6
Multiple Commands on a Single Line.....	C-7
iAPX 86 and iAPX 88 CPU Registers.....	C-8
NPX Registers.....	C-8
Errors.....	C-9
Entering Commands.....	C-9
Command Descriptions.....	C-11

FIGURES

1-1. iRMX 86 Operating System Layers.....	1-1
1-2. iRMX 86 PC Hardware Environment.....	1-2
1-3. Memory Layout of iRMX 86 System.....	1-3
1-4. Hierarchical File Structure.....	1-6
1-5. iRMX 86 PC File and Directory Structure.....	1-9
2-1. Using the iRMX 86 Operating System from a Terminal.....	2-1
2-2. DATE and TIME Commands.....	2-9
2-3. DIR, Default Format.....	2-11
2-4. SHORT, ONE-Column Directory Display.....	2-12
2-5. Display of System Directory.....	2-13
2-6. SYSTEM/UDI Directory.....	2-14
2-7. Copying a File into the Same Directory.....	2-15
2-8. Copying Multiple Files with One Command.....	2-15
2-9. Copying One File OVER Another.....	2-16
2-10. Copying a File TO an Existing File.....	2-16
2-11. Creating a New Directory.....	2-17
2-12. Displaying Contents of a File on a Terminal.....	2-18
2-13. Renaming a File.....	2-18
3-1. EXTENDED Directory Listing Example.....	3-29
3-2. FAST Directory Listing Example.....	3-29
3-3. LONG Directory Listing Example.....	3-30
3-4. SHORT Directory Listing Example.....	3-30
4-1. Chronology of System Calls.....	4-5
5-1. The iRMX 86 PC Hardware.....	5-1
A-1. Exception Code Ranges.....	A-1
A-2. iRMX 86 Condition Codes.....	A-2

CONTENTS (continued)

	PAGE
TABLES	
3-1. iRMX 86 Command Dictionary.....	3-3
3-2. Directory Listing Headings.....	3-31
4-1. System Call Dictionary.....	4-7
4-2. Command Parsing Example.....	4-21
5-1. iSBC 208 Physical Names.....	5-3
5-2. Line Printer Pin Assignments.....	5-4
5-3. iSBC 208 Jumpers.....	5-6
5-4. Interrupt Jumpers for iSBC 86/12A.....	5-7
5-5. Other iSBC 86/12A Jumpers.....	5-6
5-6. iSBC 86/12A Parallel Port Jumpers.....	5-8
5-7. iSBC 86/12A Switch 1.....	5-8
5-8. iSBC 86/12A Devices.....	5-9
5-9. Interrupt Jumpers for iSBC 86/14.....	5-10
5-10. iSBC 86/14 Parallel Port Jumpers.....	5-10
5-11. Other iSBC 86/14 Jumpers.....	5-11
5-12. iSBC 86/14 On-Board Devices.....	5-11
5-13. Interrupt Jumpers for iSBC 86/30.....	5-12
5-14. iSBC 86/30 Parallel Port Jumpers.....	5-12
5-15. Other iSBC 86/30 Jumpers.....	5-13
5-16. iSBC 86/30 On-Board Devices.....	5-13
A-1. Exception Code Ranges.....	A-1
A-2. iRMX 86 Condition Codes.....	A-2
C-1. NPX Data Types.....	C-4
C-2. iAPX 86, 88 CPU Registers.....	C-8
C-3. NPX Registers.....	C-9
C-4. Summary of Loader And Monitor Commands.....	C-11



CHAPTER 1. SYSTEM OVERVIEW

The iRMX 86 Operating System manages and extends the resources of iSBC 86 Single Board Computers. Figure 1-1 shows the structure of the Operating System; the "layers" of the system are described in Appendix B. The iRMX 86 PC Operating System -- a version of the general, configurable Operating System -- is specifically designed to allow you to develop and run programs. The features of the Operating System that are described in this chapter are:

- Support for language translators and utilities, including a standard software interface that simplifies addition of software packages to your system
- The iRMX 86 file system, including file utilities and system calls to manipulate files
- Mechanisms to bootstrap load the Operating System and to load and run programs
- Error-handling procedures

Since you will have to prepare the hardware on which the Operating System runs, the first section describes this hardware environment.

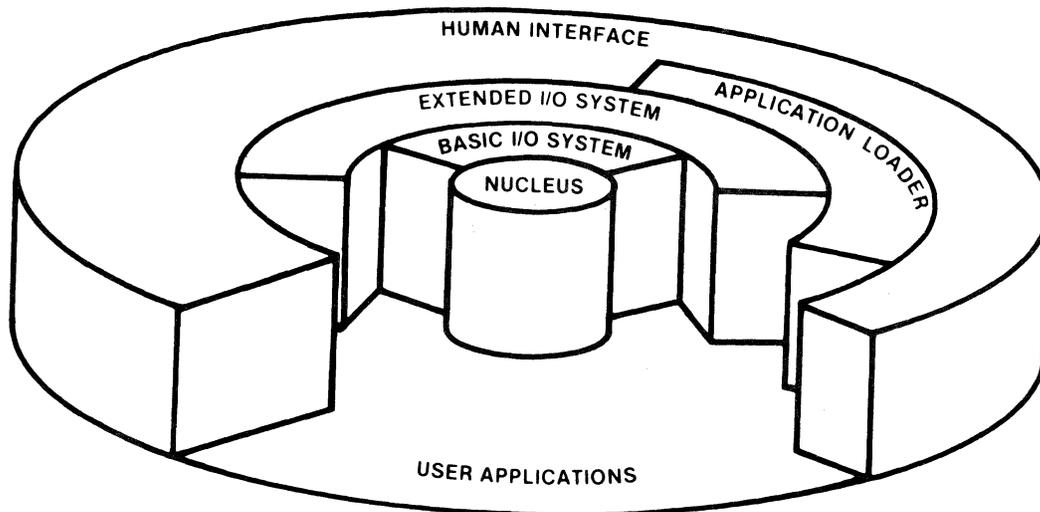


Figure 1-1. iRMX™ 86 Operating System Layers

SYSTEM OVERVIEW

HARDWARE ENVIRONMENT FOR THE iRMX 86 PC SYSTEM

The iRMX 86 PC System software is already configured for you. This section describes the hardware on which you will install the Operating System. Figure 1-2 shows the hardware, and Figure 1-3 shows a memory layout. Chapter 5, HOW TO PREPARE YOUR HARDWARE, is a self-contained guide to setting up this hardware.

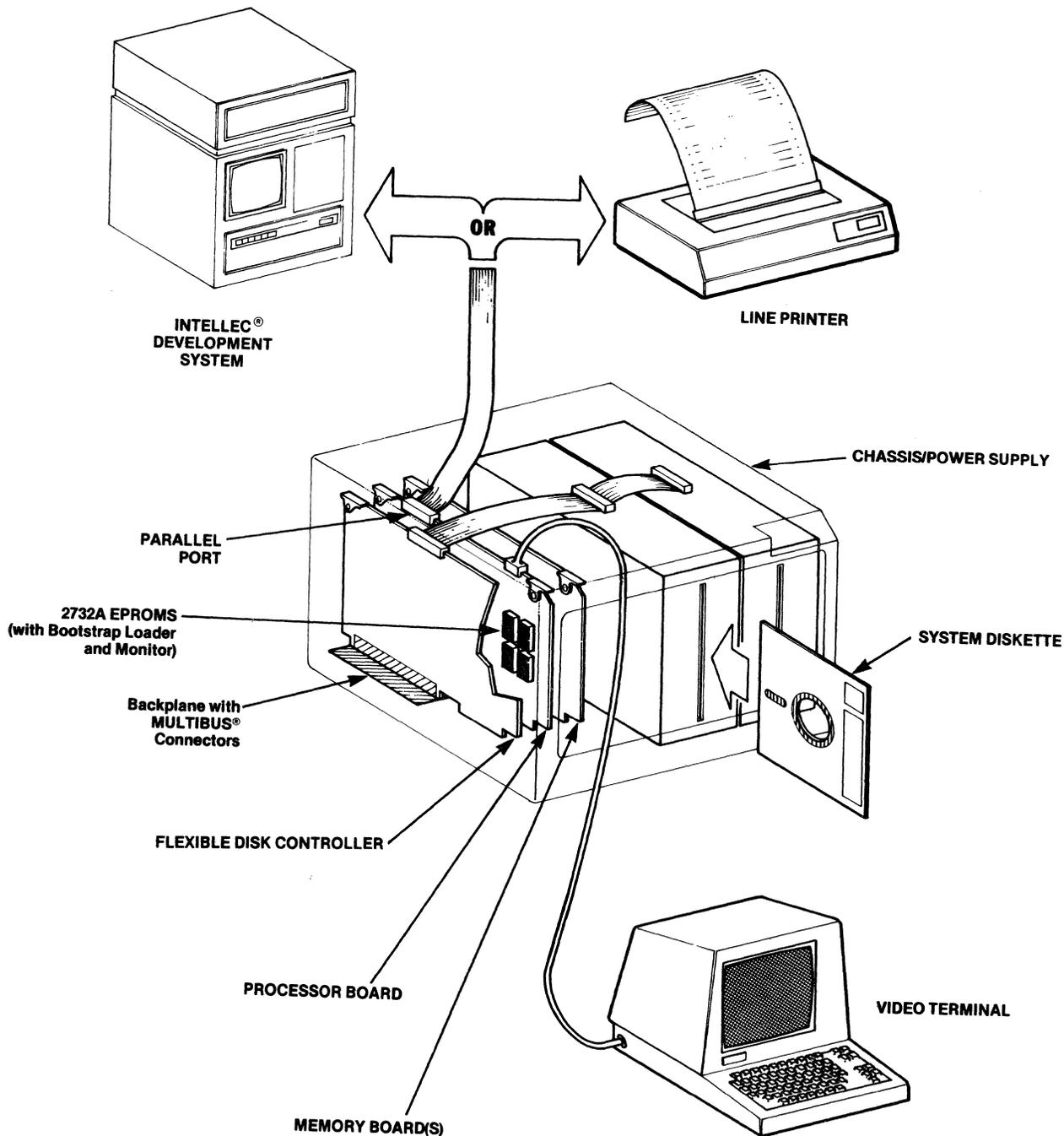


Figure 1-2. iRMX™ 86 PC Hardware Environment

SYSTEM OVERVIEW

To use the iRMX 86 PC Operating System, you require the following hardware components:

- an Intel iSBC 86/12A, iSBC 86/14, or iSBC 86/30 Single Board Computer
- an iSBC 208 diskette controller with at least two drives (you can connect as many as four drives)
- a video terminal
- an appropriate chassis/power-supply unit

In addition, you can connect either a line printer or an iSBC 957B package to the parallel port on the computer board. The iSBC 957B package allows you to connect your system directly to an Intel Microprocessor Development System. Neither the line printer nor the iSBC 957B package is required to run the Operating System.

Figure 1-3 shows a memory layout. The area labelled FREE SPACE is where your programs and iRMX 86 utilities (the commands described in Chapters 2 and 3) run. The question mark (?) on the drawing indicates that it is your choice how much free space you have on your system. You will need about 32K-bytes of free space to run the iRMX 86 commands, and more memory to run Intel compilers.

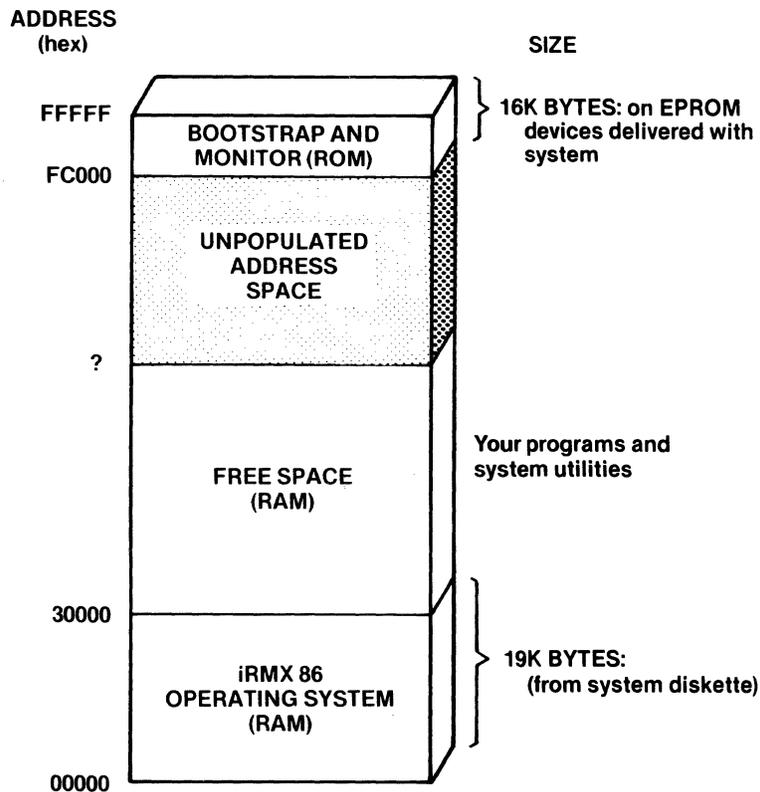


Figure 1-3. Memory Layout of iRMX™ 86 System

SYSTEM OVERVIEW

LANGUAGE TRANSLATORS AND UTILITIES

To develop programs you need language translators and utilities that allow you to compile or assemble programs, link programs together, assign absolute addresses to programs, create libraries of programs, and convert absolute object modules to hexadecimal format. Software packages available from Intel include:

EDIT	The standare iRMX 86 editor.
ASM86	The 8086/8087/8088 macro assembler.
PLM86	The PL/M-86 compiler.
LINK86	The 8086 Linker, which combines individually-compiled object modules into a single, relocatable object module.
LOC86	The 8086 Locater, which assigns absolute addresses to relocatable object modules.
LIB86	The 8086 Librarian, which creates and maintains object module libraries.
OH86	A program which converts absolute object modules to hexadecimal format.
Pascal-86	A Pascal compiler that is a strict implementation of the proposed ISO standard. It also provides extensions of the language for microcomputers.
FORTTRAN-86	A FORTRAN compiler that is compatible with existing FORTRAN-86 code, and also includes new FORTRAN-77 language features.

With these products you can create executable programs that can be invoked from the terminal. If you are an OEM (original equipment manufacturer) you can include these languages with your end product.

You can refer to the GUIDE TO USING iRMX 86 LANGUAGES for general information about invoking language products in an iRMX 86 environment. For detailed information about the software products listed here, you should refer to the manuals for the individual products. Chapter 5 contains a list of manuals assoociated with the iRMX 86 Operating System.

Contact your local Intel sales representative or distributor to order any of these software packages.

SYSTEM OVERVIEW

UNIVERSAL DEVELOPMENT INTERFACE

The iRMX 86 PC Operating System supports the Universal Development Interface (UDI). The UDI provides a standard method for your programs and for Intel software packages to use Operating System services. The UDI can be viewed as a "software bus," with the set of UDI system calls being equivalent to a hardware bus protocol. There are three important advantages to using UDI software.

- INDEPENDENCE FROM OPERATING SYSTEM CHANGES. The set of system calls for UDI remain stable regardless of changes in the Operating System, so software that you develop or install on your system can remain intact.
- PORTABILITY. With the UDI "software bus", a language translator, utility, or any other software package that uses only UDI system calls to communicate with the underlying Operating System can be installed on any operating system that supports UDI.
- INDEPENDENT VENDOR'S SOFTWARE. The UDI standard allows independent software vendors to provide a variety of programs that run on the iRMX 86 Operating System.

Chapter 4 describes how to use the UDI system calls that are available on your iRMX 86 PC Operating System.

THE iRMX 86 FILE SYSTEM

A fundamental function of the iRMX 86 Operating System is to provide a file system. Programs you write, as well as utilities and language processors, need to create and delete files; to open, close, read, and write files; and to perform other file operations. These files exist on mass storage devices such as flexible diskettes. This section describes the major characteristics of the iRMX 86 file system.

HIERARCHICAL NAMING OF FILES

People manipulate files with commands invoked from a terminal; programs manipulate files with system calls. The iRMX 86 Operating System allows your application system to organize its named files into a tree-like structure like the one shown in Figure 1-4. This hierarchical structure consists of files (represented by triangles in the figure) and directories (represented by rectangles in the figure). This hierarchy allows data to be grouped logically and accessed with a minimum of overhead.

SYSTEM OVERVIEW

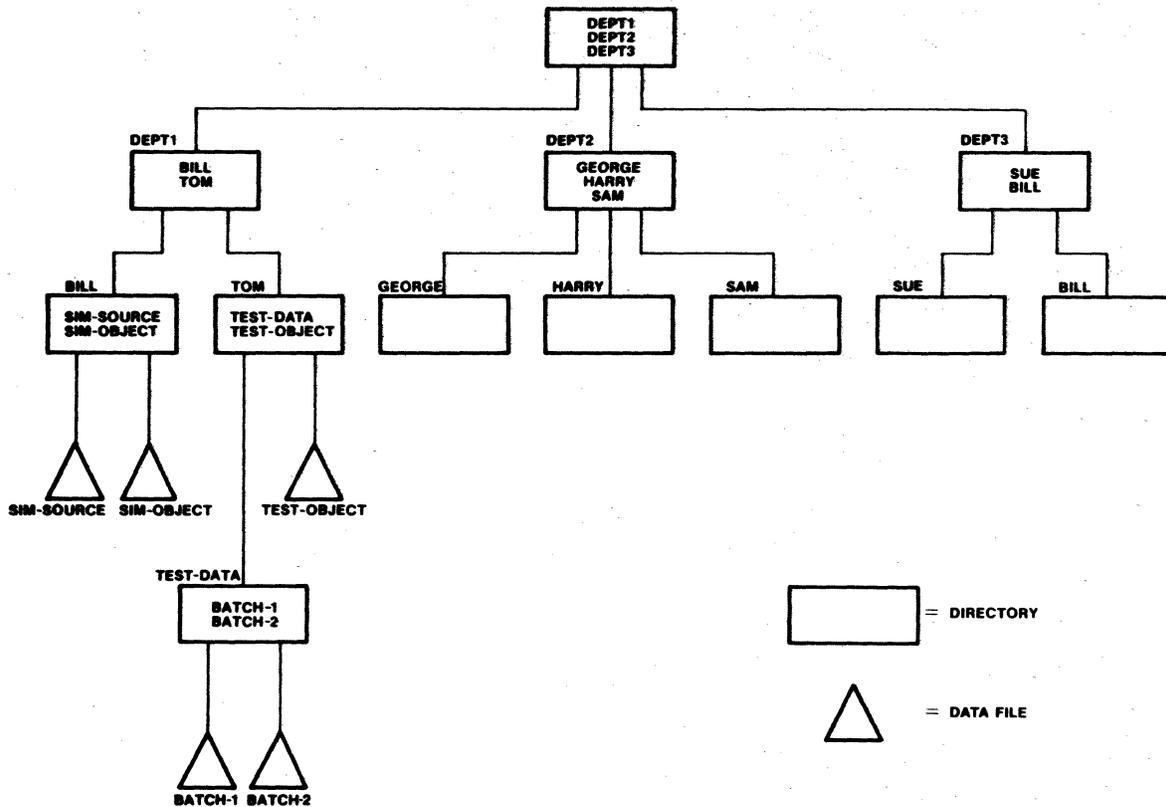


Figure 1-4. Hierarchical File Structure

iRMX 86 FILE TERMINOLOGY

Here are the meanings of terms used to describe the iRMX 86 file system:

- **File:** In a computer system, a file is simply a collection of related data known by a single name. Typically, files reside on secondary storage such as disks. An iRMX 86 file name is the last component of a pathname that implicitly or explicitly identifies the volume and directory to which the file belongs.
- **Volume:** A volume is a secondary storage device, such as a diskette, hard disk platter, or a bubble memory that you have formatted to accept files and directories. Before you can use a new volume, that volume must be formatted by using the Human Interface FORMAT command (see the FORMAT command description in Chapter 3).

SYSTEM OVERVIEW

- Directory: A directory is used to catalog (that is, to logically group and locate) files and other directories. You do not directly place information in directories as you do in a file. Rather, the Operating System maintains the directory information, adding, deleting, or changing directory entries as you add, delete, or rename files that are cataloged in the directory. Like files, directories are identified by pathnames. Files and directories contained within a particular directory must be on the same volume as the directory.
- Pathname: Normally, both system calls and commands identify a file or directory with a pathname. Pathnames can be viewed as the exact name of a file or directory. A pathname describes one path through the directory tree, so that it specifies not only the name of a file or directory, but also the particular directories and device to which it belongs.

Referring again to Figure 1-4, the name "BILL" is not exact, because there are two directories named Bill. But the pathname DEPT1/BILL is valid because it is unique. (Slash (/) characters separate the elements of a pathname.) The pathname DEPT1/TOM/TEST-OBJECT is a valid identifier for a file.

- Logical Name: Logical names are used to identify devices, and are delimited by colons (:); for example, :LP:. You assign logical names to physical devices with the ATTACHDEVICE command (see Chapter 3). Logical names that the iRMX 86 PC Operating System has already assigned to devices are described in the next section.

The reason logical names are used to identify devices is so that the same name can be used for a device even if the device is changed. For example, the iRMX 86 PC System is delivered on a single-sided, double-density diskette, and it is assumed that at the first two drives on your system have this characteristic. But it is possible to add one or two more drives, or to use the same drives with other physical characteristics (if you do the latter, you will first have to create a System Diskette in the new format). In this case you attach each drive (using the ATTACHDEVICE command) with a physical name appropriate to the characteristics of the drive. These name are listed in Chapter 5, PREPARING YOUR HARDWARE.

DEVICE LOGICAL NAMES

The logical names for devices that are used with the iRMX 86 PC Operating System are

:CI: and "Console Input" and "Console Output". Logical devices that establish the source of commands and the command destination, respectively. Typically, :CI: is the terminal keyboard and :CO: is the terminal screen.

SYSTEM OVERVIEW

:LP: "Line Printer". This is the logical name for the line printer.

:BB: "Byte-Bucket". This is not an actual physical device. Anything written to :BB: disappears, and a read from :BB: returns an EOF (end-of-file).

:AFD0: Logical names for the iSBC 208 flexible disk drives 0 and 1.

:AFD1:

FILE OPERATIONS FROM A TERMINAL

You manipulate iRMX 86 files and directories interactively with programs run from a terminal; the programs are invoked as single word commands followed by parameters. The iRMX 86 Operating System provides programs to perform operations that are usually necessary in a development system. These include:

- COPY, which copies files
- DIR, which displays the contents of a particular directory
- RENAME, which gives a new name to a file or directory
- CREATEDIR, which creates a new file directory
- SUBMIT, which automatically executes other commands contained in a file

Chapter 2 describes how to invoke commands and shows examples of some commands. Chapter 3 describes all of the commands.

FILE OPERATIONS FROM PROGRAMS

Programs must be able to manipulate files. An assembler, for example, must open and read source files, and it must create and write object files. Programs that you write will read, write, delete, and otherwise deal with files. Your programs perform these operations by means of UDI system calls. In addition to file operations, other Operating System services are available with UDI system calls. Chapter 4 contains the information you need to use UDI system calls, including individual descriptions of each call that is provided with the iRMX 86 PC Operating System. The chapter ends with a listing of a program that uses many of the UDI system calls.

SYSTEM OVERVIEW

FILES AND DIRECTORIES SUPPLIED WITH THE iRMX 86 PC OPERATING SYSTEM

The iRMX 86 PC Package contains two diskettes, a System Diskette, and a Library Diskette. The identifying labels on the outside of each are specified in the Preface.

SYSTEM DISKETTE

Figure 1-5 shows the file structure of the iRMX 86 PC System Diskette as it is delivered from Intel. The elements of this file structure are explained below.

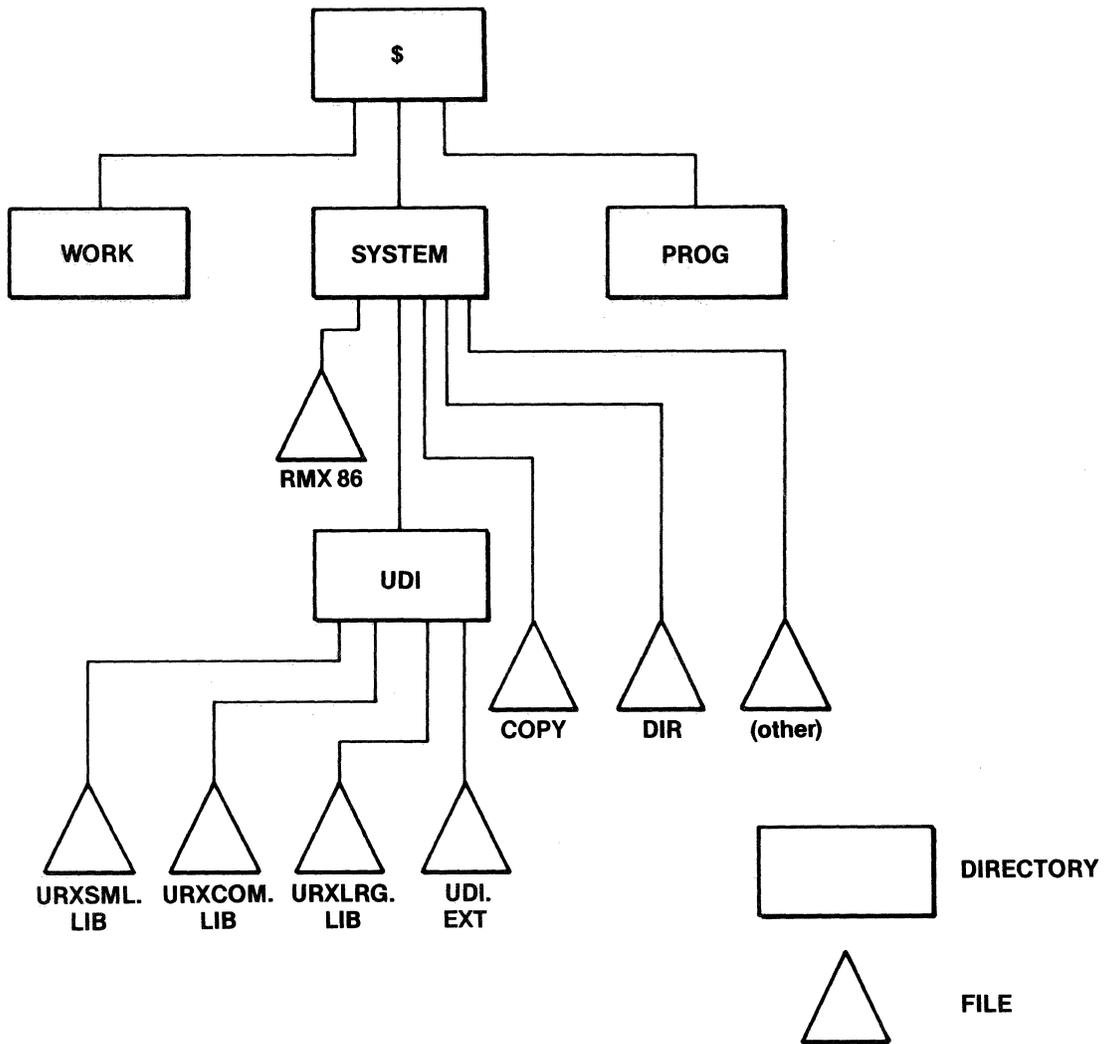


Figure 1-5. iRMX™ 86 PC File and Directory Structure

SYSTEM OVERVIEW

Default Directory (\$)

The uppermost (root) directory on Drive 0 is the default directory. If you do not specify a directory, the system will assume you are referring to this one. The root directory of a volume contains all other directories that are on the volume.

System Directory (SYSTEM)

This directory contains the following directories and files:

- IRMX 86 COMMAND PROGRAMS. When you type an iRMX 86 Command at a terminal, one of the programs in this directory is loaded and run. For example, the command "COPY" runs the program of the same name. Only a few representative command files are shown in Figure 2-4. There are 17 commands delivered with the system.
- OPERATING SYSTEM. The file RMX86 contains the iRMX86 PC Operating System; this is the file that is read in by the Bootstrap Loader (see BOOTSTRAP LOADING in a later section).
- UDI LIBRARIES. The directory UDI contains three library files that allow programs to use the UDI system calls. When you use a language processor like the PL/M-86 compiler to write a program, you link the resultant object modules to one of these libraries. This is explained more fully in Chapter 4.

Program Directory (PROG)

You can use this directory for programs that you write, and you can create other directories within this one to provide a logical grouping of your files.

The PROG directory has one special characteristic. When you type a single-word command at a terminal, the iRMX 86 Operating System will first look for the program file with that name in the default directory, then in the PROG directory, and finally in the SYSTEM directory. Two effects of this are:

1. Programs in the PROG directory will be executed as commands when you simply type the single-word file name. For example, if you have a program file PROG/UPPER, you can run the program by simply typing the command UPPER.
2. If you have a program named, for example, "COPY" in the PROG directory, when you type "COPY" the system runs your program rather than the COPY program supplied by Intel.

If a program file is contained in a directory other than one of these three (\$, PROG, or SYSTEM), you can still run the program by typing its complete pathname.

SYSTEM OVERVIEW

Work Directory (WORK)

Compilers, interpreters, editors, linkers, and other development utilities need to create temporary files while they are running. This directory is specifically provided for that use.

LIBRARY DISKETTE

Besides the System Diskette, you receive another diskette with the iRMX 86 PC product, labelled "iRMX 86 Interface Libraries." This diskette contains:

- Libraries that you need if you use iRMX 86 System Calls not described in Chapter 4 (these calls are listed in Appendix B).
- Files of symbolic names for exception codes (listed in Appendix A).
- Files of external declarations associated with each layer of the Operating System (the layers are briefly described in Appendix B).

The Library Diskette is not required for normal program development. If you use these files, you will probably need one or more of the manuals listed in Chapter 6.

PROGRAM LOADING

Bringing programs into memory (loading) from the disk is one of the basic services provided by the iRMX 86 Operating System; you load and run language processors, utilities, and the programs you write. Typically, you load programs by simply typing the single-word name of a program file at a terminal, sometimes followed by other information needed by the program when it begins executing.

BOOTSTRAP LOADING

To get the iRMX 86 PC Operating System into the computer from disk, the system is bootstrap loaded. This process is described in Chapter 2. The Bootstrap Loader is in the set of EPROM devices delivered with the iRMX 86 PC Package.

SYSTEM OVERVIEW

MONITOR

Your iRMX 86 PC System is delivered with a Monitor. Like the Bootstrap Loader, the Monitor is in EPROM devices you receive with the iRMX 86 PC Package. It can be used to examine memory, set breakpoints, and (with a hardware package available separately) to communicate between your system and an Intellec Development System.

Monitor commands are described in Appendix C.

SELECTIVE ERROR PROCESSING

When a program issues an iRMX 86 system call, the results may be other than what the programmer expected. For example, a program might request memory that is not available, or it might use an invalid parameter. The iRMX 86 PC Operating System contains a default exception handler that will terminate a program if such a condition occurs; the default exception handler will identify the problem by displaying on the console terminal one of the exception codes listed in Appendix A.

If you want to provide your own exception handler, rather than using the default exception handler, the Operating System provides a mechanism for transferring control to your exception handler. The system calls used to write an exception handler are described in Chapter 4, UDI SYSTEM CALLS.

SUMMARY

The iRMX 86 Operating System is a flexible operating system that is used for many types of systems. This chapter has discussed only those features that directly relate to using the Configured iRMX 86 Operating System for program development. For more complete discussions of iRMX 86 Operating System features, refer to the INTRODUCTION TO THE iRMX 86 OPERATING SYSTEM.

CHAPTER 2. USING THE SYSTEM

You communicate with the iRMX 86 Operating System by using commands entered at a terminal keyboard (Figure 2-1); the Operating System communicates with you by displaying messages on the terminal screen. This chapter describes the process of using the Operating System, showing some examples of iRMX 86 commands and system responses. Chapter 3 describes all of the commands that Intel provides with the iRMX 86 PC Operating System.

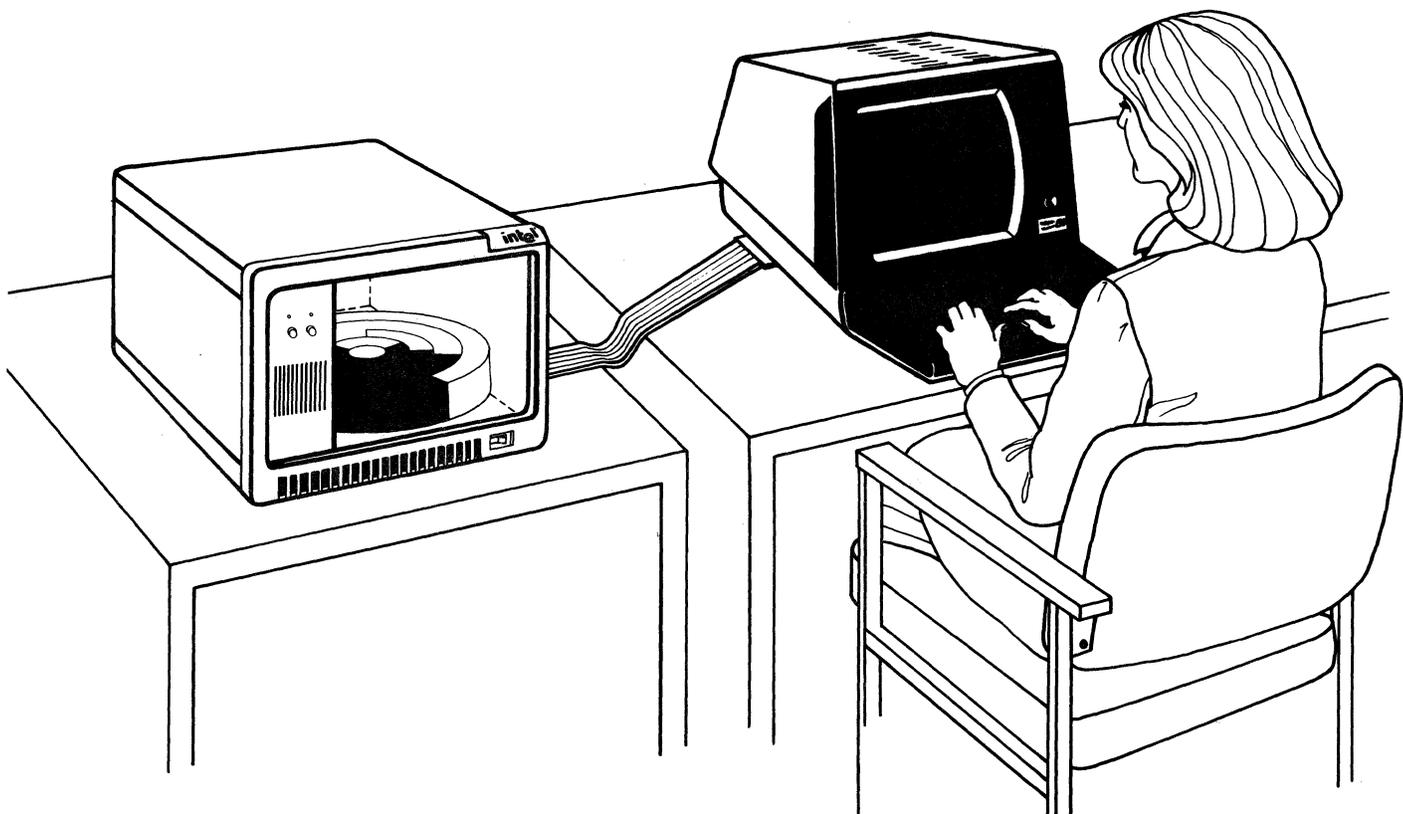


Figure 2-1. Using The iRMX™ 86 Operating System From A Terminal

USING THE SYSTEM

The chapter is organized as follows:

- STARTING THE SYSTEM. A section showing how to start (bootstrap load) the system.
- INVOKING iRMX 86 COMMANDS. General information including definition of terms used to describe individual commands.
- EXAMPLE COMMANDS. A section showing examples of iRMX 86 commands, most of which manipulate files.

STARTING THE SYSTEM

Once you have prepared your iAPX 86,88-based hardware, as described in Chapter 5, you can bootstrap load ("boot") the Operating System. Bootstrap loading is the process of reading the iRMX 86 Operating System in from a disk and giving it control of the processor. Here is how to boot the system.

1. Turn on power to the disk drive, processor, and terminal.
2. Insert a copy of the System Diskette into Disk Drive 0. (You should make a copy of the diskette that you receive from Intel, and use the copy rather than the original diskette. How to do so is explained at the end of this chapter.)
3. When the terminal shows a series of "*" (asterisk) characters, respond by typing an upper-case "U". (The system continues sending asterisks to the screen until you type a "U." The "U" is not echoed on the screen.)
4. The terminal shows a message identifying the Monitor, followed on the next line by a prompt of "." (period):

```
iAPX 86, 88 Monitor, V1.0
```

```
.
```

5. You respond by typing the single character "B" (upper- or lower-case) followed by a carriage return (CR).
6. Now the Bootstrap Loader reads the Operating System into memory from your diskette, and passes control to it. (This takes about one-half minute.)
7. The Operating System displays a message identifying itself, followed on the next line by a prompt of "-" (hyphen):

```
iRMX 86 PC V1.0: user = WORLD
```

```
-
```

At this point the system is loaded and you can enter any iRMX 86 command.

USING THE SYSTEM

If your system has a button connected to the RESET line on the iSBC 86 board, you can use it to re-boot: after hitting RESET the system will begin displaying astericks (*) on the screen, and you continue from step 3 above.

The command DEBUG can be used to get to the Monitor, and from the Monitor you can also re-boot the system. See the description of DEBUG in Chapter 3, and the Monitor commands described in Appendix C.

CAUTION

To prevent destroying data on your diskettes while re-booting, wait at least 2 seconds before you RESET the computer.

INVOKING iRMX 86 COMMANDS

This section describes procedures and defines terms that apply to iRMX 86 commands. Examples of actual commands are shown in the next section, and additional information about individual commands is in Chapter 3.

When you enter a command at the console keyboard, the Operating System loads the associated program file and executes the program. After the command has been executed, the Operating System displays a status message that confirms the effect of the command.

The Operating System displays error messages if you attempt an invalid operation (e.g., trying to access a file that doesn't exist) or if some error is encountered while the command is being executed (such as a hardware failure). These error messages are defined as part of individual command descriptions in Chapter 3.

USING THE SYSTEM

An example command is:

```
-COPY first, second TO third, fourth QUERY
```

A carriage return terminates each command, and a LINE-FEED key has the same effect. From here on we will assume a carriage return is the terminator. The general structure of a command is shown next.

```
command-name inpath-list preposition outpath-list parameters
```

where:

command-name Name of the program file to be executed. After the command is entered, the Operating System loads the program file into memory from the diskette and executes the command. In the example, the command name is:

```
COPY
```

inpath-list One or more pathnames of files to be used as input during command execution. Multiple pathnames in an input file list must be separated by commas. You can type spaces (blanks) between pathnames. In the example, the inpath-list is:

```
first, second
```

preposition A word that tells the executing command how you want the output handled. The four prepositions used in IRMX 86 commands are TO, OVER, AFTER, and AS. In the example the preposition is:

```
TO
```

outpath-list One or more pathnames for the files that receive the output or are changed in some way. As with the inpath-list, multiple files names must be separated by commas, and embedded spaces are optional. In the example, the outpath-list is:

```
second, third
```

parameters Most commands have have a default form, but also offer one or more optional ways that the system can execute the command. You specify options with one or more parameters at the end of a command. Individual descriptions of commands in Chapter 3 define the effect of parameters. In the previous example, the parameter is:

```
QUERY
```

USING THE SYSTEM

You can also continue a command beyond one line, and you can add comments at the end of a command:

continuation mark If you need to type a command that is so long it cannot be typed on one line, you can continue it by typing an ampersand (&) character and carriage return. The system prompts with two asterisks (**) on the next line, and you can then continue the command.

You can continue the command for as many lines as are necessary. A carriage return key without an ampersand ends the command line. A command line can have a up to 255 characters, including punctuation, embedded blanks, continuation mark, comments, and carriage return.

comment A semicolon (;) character causes the system to ignore anything typed between the semicolon and the succeeding carriage return. You can also type comments between a continuation mark (&) and the carriage return. A common use of comments is in SUBMIT files (see the SUBMIT command in Chapter 3).

You can type all elements of a command in uppercase characters, lowercase characters, or a mix of both. For example, you can create a new file with the pathname "MY/TEST" and then specify the file as "my/test" in subsequent file accesses.

PREPOSITION PARAMETERS

Most file management commands recognize three prepositions: TO, OVER, and AFTER. (The preposition AS is used in the ATTACHDEVICE command and is explained with that command in Chapter 3.) The prepositions have the following meaning:

TO Causes the command to send the output to new files; that is, to files that do not already exist in the specified directory. If the output file does exist, the command will display the following message on the console screen:

pathname, already exists, DELETE?

In general, you can reply "Y" (for yes) if you accept destroying the output file contents. Usually any other character means "no", but there are some exceptions to this. Check individual command descriptions in Chapter 3.

USING THE SYSTEM

OVER Causes the command to replace the contents of files specified in the outpath-list with the contents of the input files, destroying the contents of the output files. For example:

```
-COPY samp1,samp2 OVER out1,out2
```

copies the data from file samp1 over the present contents of file out1, and copies the data of samp2 over the contents of file out2. If either out1 or out2 did not exist, the file would be created. Neither input file changes.

AFTER Causes a command to append the contents of one or more files to the end of new or existing files. For example:

```
-COPY in1,in2 AFTER dest1,dest2
```

causes the contents of file in1 to be written to the end of the contents of dest1, and the contents of in2 to be added to the end of dest2. (Neither in1 or in2 change in any way.)

TERMINAL CONTROLS

Certain keys at the terminal have special effects on the Operating System. These are listed and described here.

NOTE

In this manual, CONTROL key functions are designated as follows:

CTRL/character

where CTRL specifies the CONTROL key, and character is an alphabetic character key. Depress the CTRL key while striking the letter key.

CTRL/c Tells the iRMX 86 Operating System to abort the currently executing program.

CTRL/o Suppresses terminal output, or restores output to normal mode if output is already suppressed. Typically this is used to ignore ("throw away") data being sent to a terminal.

USING THE SYSTEM

CTRL/s CTRL/q	Suspends and resumes output to the terminal. Unlike using CTRL/o, output is not ignored; the system stops sending output to the terminal until you press CTRL/q. When you press CTRL/q, you see the remaining output.
CTRL/r	Repeats the current line so that you can modify it before the command is executed. If the line is empty, the system echoes the previous command so that you can re-execute it.
CTRL/x	This is used to delete a currently displayed command line and allows you to start the command again. The Operating System will echo a pound sign (#) at the point where you strike CTRL/x, and then move the screen cursor to the beginning of the next line.
CTRL/z	This an End-of-File character for the Console Input device; if you use it, it should be entered as the first character in a new line.
RUBOUT	Permits simple editing on the current line. Each time the RUBOUT key is pressed, the last displayed character is deleted with the cursor moving backward one space. You continue pressing the RUBOUT key until you reach the character to be corrected.

In Chapter 4, the description of DQ\$SPECIAL includes a definition of "transparent mode" input from the :CI: device, in which characters described here do not have their normal effect.

UNEQUAL NUMBER OF FILES IN INPUT AND OUTPUT LISTS

Several iRMX 86 commands require that you specify a preposition parameter in the command. That is, you must enter a TO, OVER, or AFTER preposition as one of the command parameters. Usually you specify a one-for-one match between the number of input files and number of output files. (This is a requirement for the command RENAME.) But the following sections explain what happens when the number of files specified in the inpath-list does not equal the number of files in the outpath-list.

More Input Files Than Output Files

In a command (other than RENAME), if you specify more pathnames in the inpath-list than in the outpath-list, the remaining input files are automatically appended to the end of the last specified output file, regardless of the preposition you specified. For instance, assume that in a COPY command you specify the following file names in the input and output parameters:

```
COPY a,b,c TO d,e
```

USING THE SYSTEM

When the Operating System executes the command, file "a" is copied to file "d", file "b" is copied to "e", and file "c" is appended to the end of file "e" as follows:

```
a TO d
b TO e
c AFTER e
```

More Output Files than Input Files

If you specify more file names in the outpath-list than in the inpath-list, the excess output file names are ignored, again regardless of the preposition you specify. For example, assume that in a command you specify the following file names in the input and output parameters:

```
COPY a,b TO d,e,f,g
```

When the command is executed, file "a" is be matched with file "d", file "b" copied to file "e", and files "f" and "g" are ignored, as follows:

```
a TO d
b TO e
```

Safeguards

A mismatch between the number of input files and output files is probably accidental. The iRMX 86 Operating System attempts to execute commands without destroying the integrity of your files. When the Operating System encounters a command that is subject to ambiguous interpretation or could result in the accidental destruction of an existing file, the command displays a message and prompts you to confirm or cancel the operation.

EXAMPLE COMMANDS

This section shows some examples of iRMX 86 commands. These examples are deliberately few and simple. The examples demonstrate some representative commands so that you can see how to invoke a command and how to specify command parameters. Once you are familiar with the process of invoking commands with some typical pathnames and parameters, refer to the complete descriptions of commands in Chapter 3 (many of which also include examples).

HOW TO SET THE SYSTEM DATE AND TIME

Two of the easiest commands to use are DATE and TIME. They are shown first because it is good practice to set the system date and time immediately after the system has been bootstrap loaded. Figure 2-2 shows how to use both commands.

USING THE SYSTEM

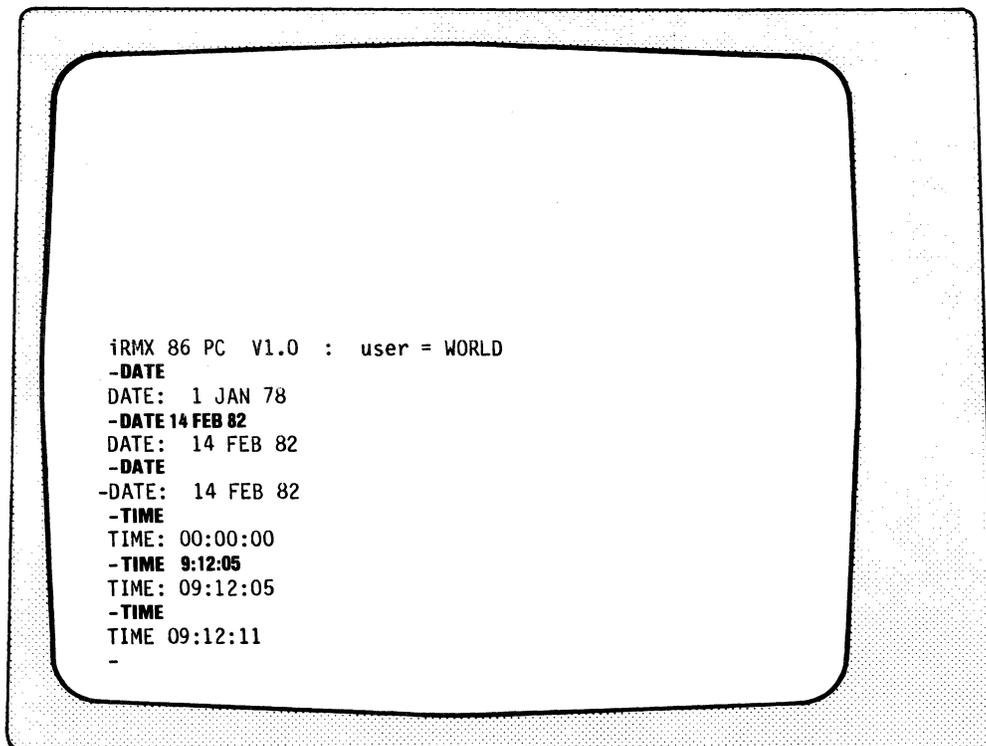
In Figure 2-2, you see the message displayed after the Operating System has been booted, followed by a hyphen prompt (-). This is exactly where we left the system in the first section of this chapter, STARTING THE SYSTEM. The DATE command typed in response to the first prompt displays an arbitrary date that indicates it has not been set since the system was booted.

NOTE

In examples of terminal dialogue, commands that you type are shown in THIS TYPEFACE.

Messages displayed by the system are shown in THIS TYPEFACE.

The first examples show a full screen, but later illustrations show only the lower portion of the screen. Illustrations are not proportional to an actual video screen.



```
iRMX 86 PC V1.0 : user = WORLD
-DATE
DATE: 1 JAN 78
-DATE 14 FEB 82
DATE: 14 FEB 82
-DATE
-DATE: 14 FEB 82
-TIME
TIME: 00:00:00
-TIME 9:12:05
TIME: 09:12:05
-TIME
TIME 09:12:11
-
```

Figure 2-2. DATE And TIME Commands

USING THE SYSTEM

The fourth line in the Figure 2-2 shows the date being set to Valentines Day of 1982. The system then responds by verifying the new date. The next line illustrates that any time DATE is typed without specifying a date, the current date is displayed.

The next lines show the same sequence for the TIME command. The system first responds by displaying the system time as

```
00:00:00
```

Next the time is set to 12 minutes and 5 seconds after 9 AM, and the system verifies it on the next line. Finally, the TIME command re-typed shows the updated time.

If you don't set the system time or date, the iRMX 86 Operating System will not maintain the system clock. Two results of this are:

1. Whenever you interrogate the system to determine the time-of-day -- whether by commands as shown here, or with a programmed system call as shown in Chapter 4 -- the time will remain fixed at zero-hour:zero-minute:zero-second.
2. When you display the contents of a directory, the line showing the date and time will not be shown. (Displaying directories is the subject of the next few examples.)

HOW TO DISPLAY THE CONTENTS OF A DIRECTORY

Frequently you need to see what files -- and other directories -- are catalogued in a particular directory. This is the function of the DIR command, and a few examples are shown here.

Using the DIR Command with no Parameters

Figure 2-3 shows the effect of typing the DIR command with no pathname or parameters. The system displays:

- o The current date and time, followed on the next line by
- o A message identifying the directory, followed on the next line by
- o the name of each file and directory.

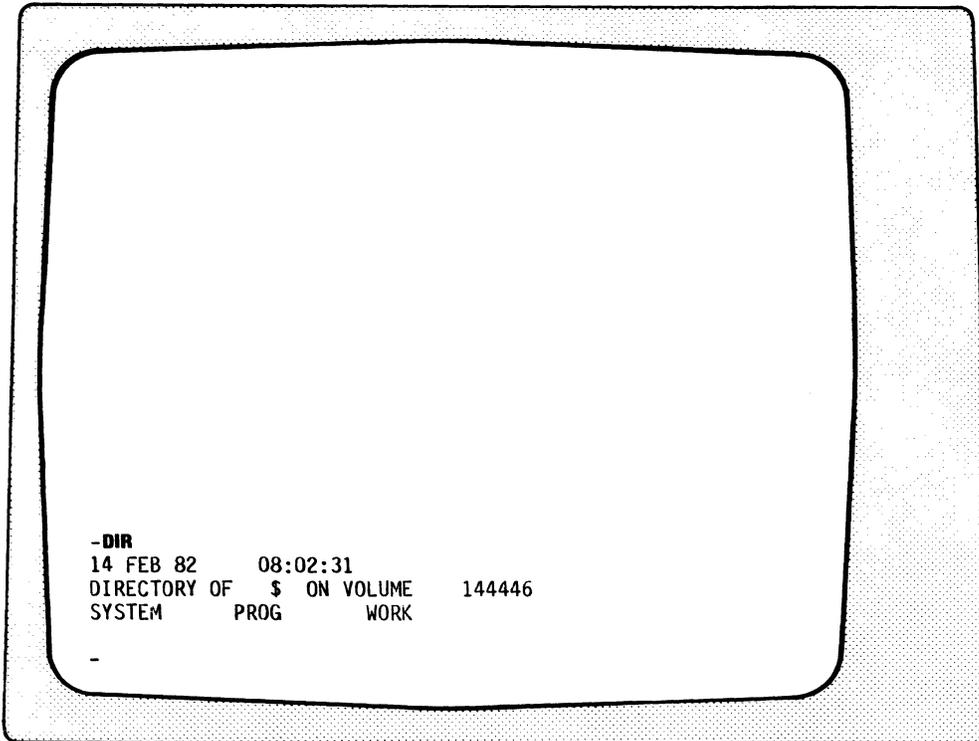


Figure 2-3. DIR, Default Format

NOTE

These examples of the DIR command assume that you are using an exact copy of the iRMX 86 PC System Diskette, that it is in Drive 0, and that no files or directories have been added or deleted. (Figure 1-5 shows the contents of this diskette.)

If you do not specify any other parameters in a DIR command, only the names of the directory entries are displayed. In the example in Figure 2-3, you cannot tell the size of each entry, or whether it is a directory or a file. By specifying an optional display format with a DIR command, you can see these and other characteristics of the directory entries. DIR offers you a variety of display options, one of which is shown in the next example.

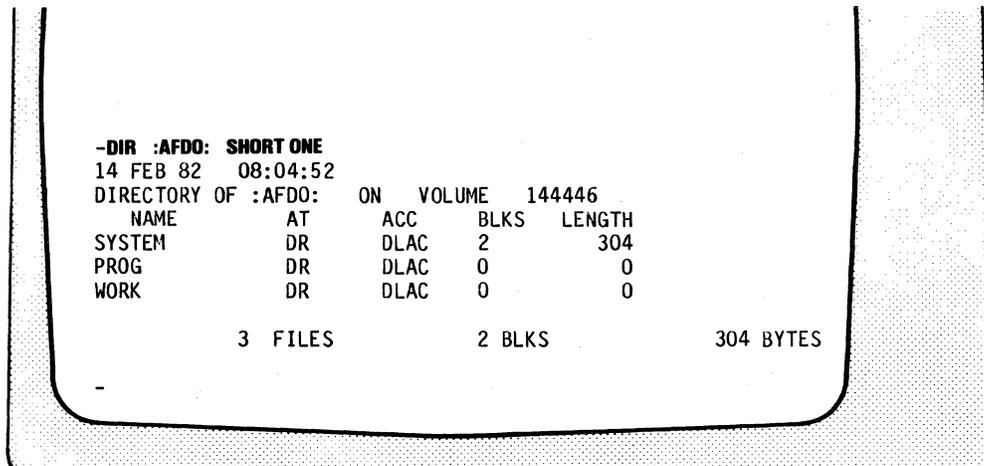
The VOLUME number 144446 is the name by which the system knows the particular diskette on which the directory is located. This name or number may vary; it is established when a disk is initialized (see the FORMAT command in Chapter 3).

USING THE SYSTEM

If you do not specify a device, the system will assume that you are using Disk Drive 0 (zero). This is known as the default system device. The uppermost (root) directory on Drive 0 is the default directory (\$); this is true regardless of what diskette you have in that drive. If a file or directory is on drive 0, you do not usually have to specify the drive (an exception is shown later). If a file or directory is in the default directory you don't have to specify a directory name.

Directory Displayed in an Alternate Format

Figure 2-3 again shows the contents of the default directory, but with the parameters SHORT and ONE. With optional parameters, you can control the physical format (ONE specifies that only one entry be on a line) and the type of information displayed (SHORT displays more information than in the previous example).



```
-DIR :AFDO: SHORT ONE
14 FEB 82 08:04:52
DIRECTORY OF :AFDO: ON VOLUME 144446
  NAME      AT      ACC      BLKS      LENGTH
SYSTEM     DR      DLAC      2         304
PROG       DR      DLAC      0          0
WORK       DR      DLAC      0          0

          3 FILES          2 BLKS          304 BYTES
```

Figure 2-4. SHORT, ONE-Column Directory Display

Figure 2-4 shows, in addition to the names of each entry:

- Whether the entry is a file or directory (AT column, where DR means directory)
- The access rights (ACC column)
- The number of blocks and number of bytes (BLKS and LENGTH)
- A summary of everything in the directory

USING THE SYSTEM

One further comment about Figure 2-4; the drive (:AFDO:) is specified even though it is the default system device. This is because if the command had been typed without the drive number -- DIR SHORT ONE -- the Operating System would have looked for a directory named SHORT. You may wonder what happens when you type a command that the Operating System cannot interpret correctly. In this case, the system would not have found a directory with the name SHORT, and would have displayed the error message:

```
SHORT, file does not exist
```

In this case, if there had been a directory named SHORT within the default directory, the DIR command would have displayed its contents.

Directory Listing of SYSTEM Directory

Figure 2-5 shows the directory named SYSTEM, using the default listing format (no parameters).

```
-DIR SYSTEM
14 FEB 82 08:05:01
DIRECTORY OF SYSTEM ON VOLUME 144446
CREATEDIR BACKUP COPY RMX86 SUBMIT
DISKVERIFY DIR DELETE RENAME RESTORE
ATTACHDEVICE DOWNCOPY FORMAT TIME DATE
DETACHDEVICE UPCOPY UOI DEBUG
-
```

Figure 2-5. Display of System Directory

USING THE SYSTEM

Directory Listing of SYSTEM/UDI

In Figure 2-5, note the entry "UDI". This is the directory containing the UDI library files. The next example, Figure 2-6, displays this directory in the default style (no parameters specified) and then in a SHORT, ONE-column style.

```
-DIR system/udi
14 FEB 82 08:09:12
DIRECTORY OF system/udi ON VOLUME 144446
UDI.EXT URXCOM.LIB URXSML.LIB URXLRG.LIB

-DIR system/udi SHORT ONE
14 FEB 82 08:09:32
DIRECTORY OF system/udi ON VOLUME 144446
NAME AT ACC BLKS LENGTH
UDI.EXT DRAU 14 3426
URXCOM.LIB DRAU 140 35682
URXSML.LIB DRAU 142 36118
URXLRG.LIB DRAU 141 35888

4 FILES 437 BLKS 111114 BYTES
```

Figure 2-6. SYSTEM/UDI Directory

Note that each entry is a file, as indicated by the blank entry under AT. Also, the name of the directory was typed in lower case; the effect was the same as if it had been typed in upper case.

HOW TO COPY FILES

The next few examples show how the COPY command is used to duplicate files. For these examples, assume that a file named FIRST exists on diskette :AFDO: in the default directory.

Creating a New Copy of a File

Figure 2-7 shows how to create a single copy of a file in the same directory.

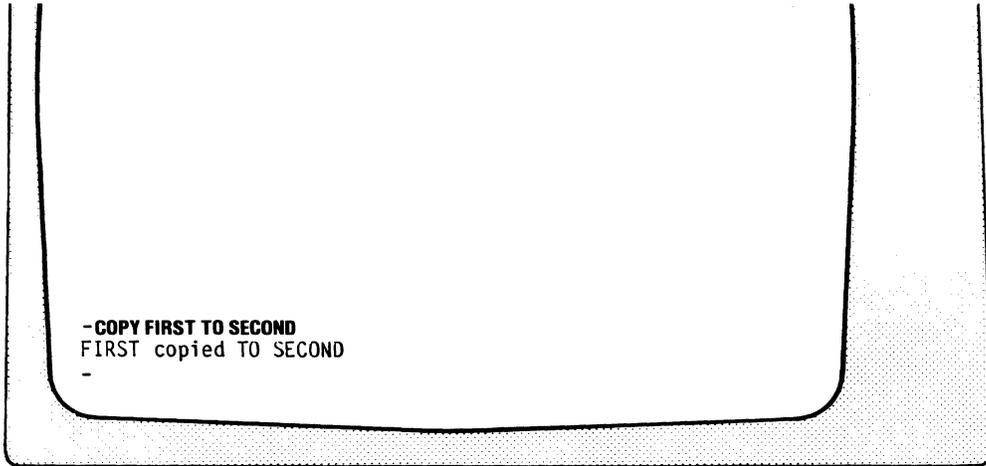


Figure 2-7. Copying a File Into the Same Directory

Copying Multiple Files With One Command

It is possible to copy more than one file with a single copy command. Figure 2-8 shows how to create copies of the files FIRST and SECOND.

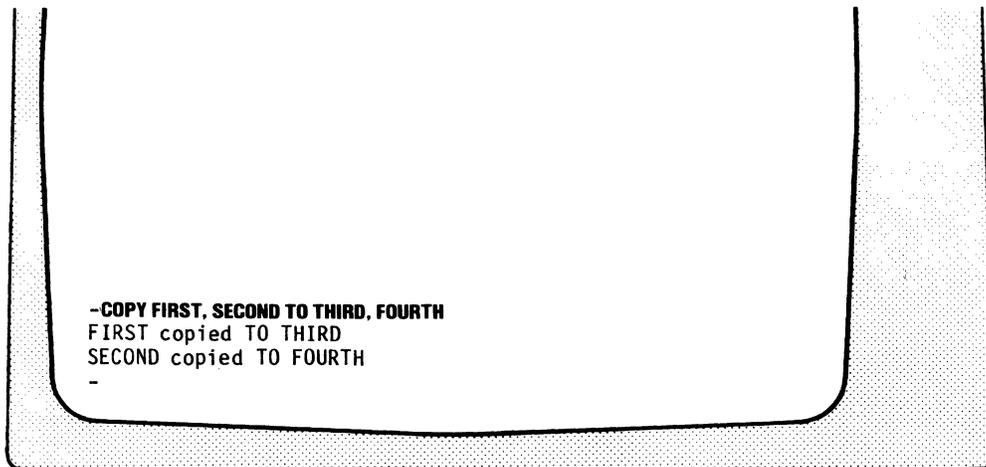


Figure 2-8. Copying Multiple Files With One Command

USING THE SYSTEM

Copying One File OVER Another

You can copy the contents of one file into another. The preposition OVER tells the system to destroy the current contents of the file specified in the outpath-list and copy the contents of the file specified in the inpath-list into the file. This is shown in Figure 2-9.

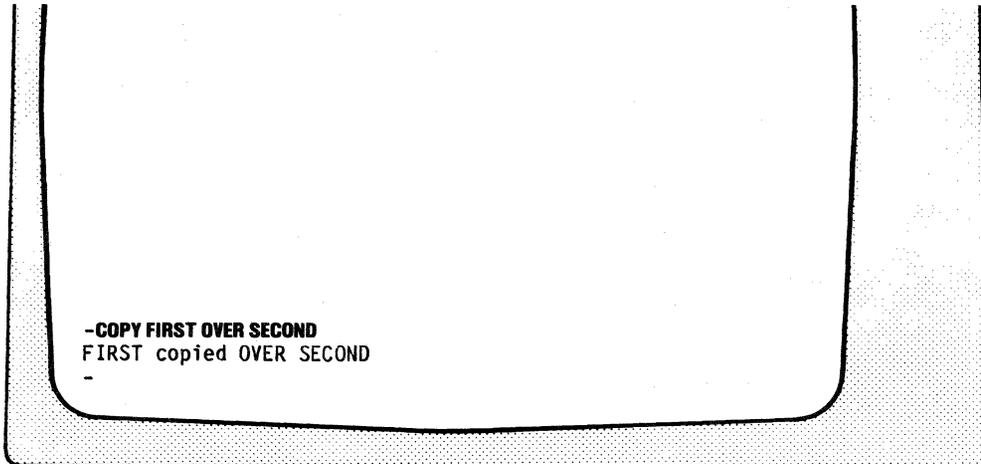


Figure 2-9. Copying One File OVER Another

Using the OVER preposition explicitly deletes the contents of a file. Using the preposition TO can have the same effect, but with one difference. If you use the TO preposition to copy a file into a file that already exists, the system displays a message asking if you actually want to destroy the contents of the existing file (as shown in Figure 2-10).

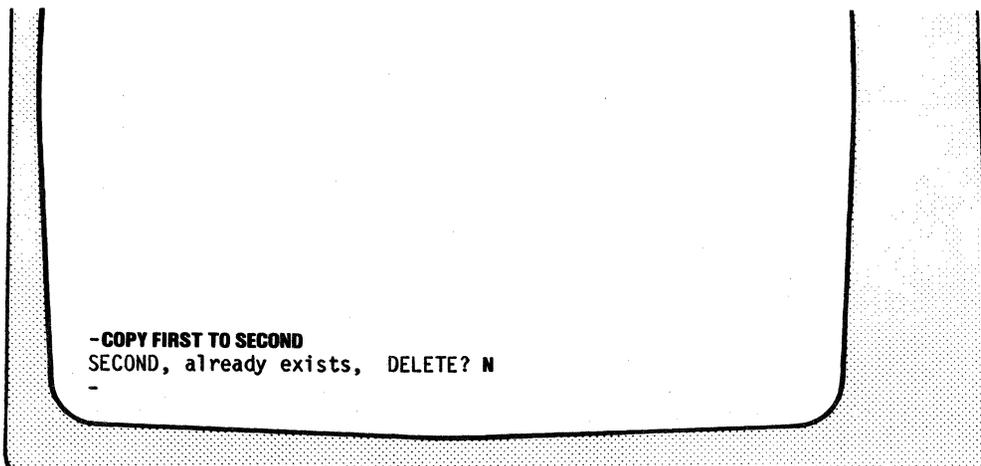


Figure 2-10. Copying a File TO an Existing File

USING THE SYSTEM

In the example in Figure 2-10, the reply to the query "DELETE?" is "N", which tells the system not to destroy the file. The operation was cancelled. If the reply to "DELETE?" had been "Y", the contents of SECOND would have been destroyed and the contents of FIRST copied into the new file SECOND.

CREATING A DIRECTORY

The CREATEDIR creates new directories. For example, you can create a directory within your default directory (\$) and then copy files into that directory. This is shown in Figure 2-11.

```
-CREATEDIR NEW
NEW, directory created
-COPY FIRST TO NEW/FIRST
FIRST copied TO NEW/FIRST
-
```

Figure 2-11. Creating A New Directory

DISPLAYING THE CONTENTS OF A FILE AT THE TERMINAL

You can view the contents of a file at the console terminal (:CO:) using the COPY command. The file should contain string data (like the output of an editor) rather than binary data (like the object file from a compiler). Displaying a binary file will produce a meaningless display, although it will not affect the file.

The Operating System assumes "TO :CO:" if you do not specify either a preposition or outpath-list with a COPY command. Figure 2-12 shows the command to display the contents of the file SECOND.

USING THE SYSTEM

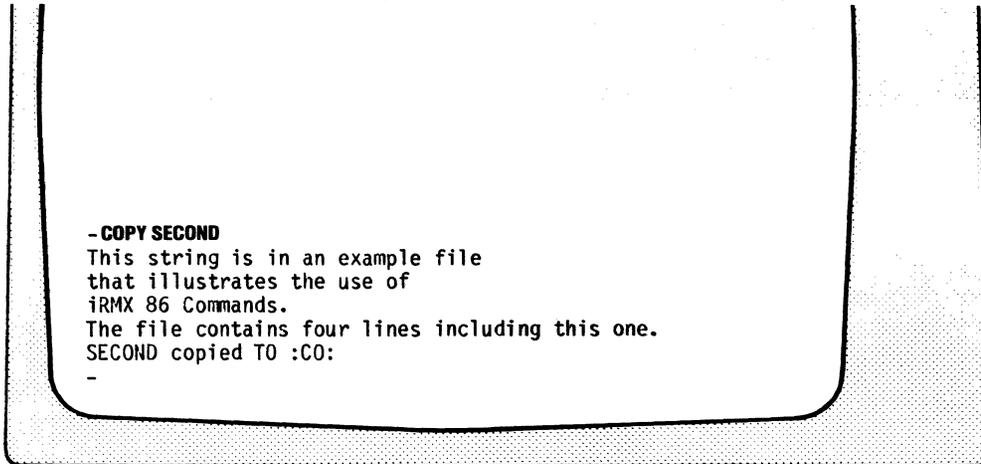


Figure 2-12. Displaying Contents of a File on a Terminal

GIVING A FILE A NEW NAME

The Operating System provides the **RENAME** command to rename a file. You can use the **COPY** and **DELETE** commands to accomplish the same thing. However, the **COPY** command actually moves the contents of the file being copied and leaves the original file intact. The **RENAME** command leaves the file intact, but changes the pathname. Figure 2-13 shows the **RENAME** command used to rename **THIRD**.

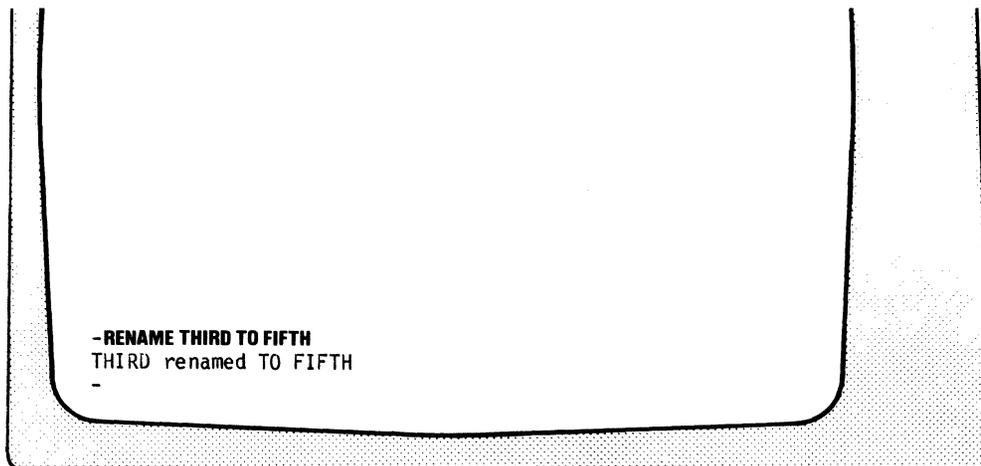


Figure 2-13. Renaming a File

Both files and directories can be renamed. If a directory is renamed, any files or directories cataloged under that directory will automatically have new pathnames. You should note this if programs you have written use files in the directory.

The **OVER** preposition is valid in a **RENAME** command, and its effect is explained in Chapter 3.

USING THE SYSTEM

HOW TO MAKE COPIES OF YOUR SYSTEM DISKETTE

You should make a copy of the System Diskette that you received with the iRMX 86 PC package. Listed below are the steps necessary to do so. We assume that you have only two disk drives on you system, and that you have read about the BACKUP, RESTORE, and FORMAT commands in Chapter 3.

To make a new Sytem Diskette:

1. Format two new diskettes, one as a PHYSICAL volume, and one as a NAMED volume. (BACKUP writes to a PHYSICAL-formatted diskette, and RESTORE copies from this diskette to a NAMED volume.) Both diskettes can be formatted in Drive 1.
2. Using BACKUP, write the System Diskette contents onto the diskette that you formatted as a PHYSICAL volume. One of the parameters to BACKUP is pathname; you should specify only the volume (:AFDO:).
3. Run RESTORE. When the RESTORE program prompts for the volume to be mounted, remove the system diskette from Drive 0, insert the volume that you formatted as a NAMED volume into Drive 0, and type Y. When invoking RESTORE, again specify :AFDO: as the pathname.
4. When RESTORE completes, you should be able to re-boot from the new diskette. Save both the diskette you received from Intel and the diskette which BACKUP wrote.

NOTE

When formatting your diskette, specify an interleave factor of seven (7) rather than the FORMAT command default value of five (5). The reason is that an interleave factor of five will result in a much slower bootstrap process: nearly two minutes rather than about one-half minute.

CHAPTER 3. iRMX™ 86 COMMANDS

The commands described in this chapter are supplied by Intel. You can use the commands to perform a number of highly convenient file management functions. When you invoke a command,

1. You type the command name and parameters (e.g., "COPY FIRST TO SECOND").
2. The the Operating System loads the the appropriate command file (for example, SYSTEM/COPY) and executes the program.
3. The program executes the command the way that you specify in the command line.

These commands are part of the iRMX 86 Human Interface (one layer in the Operating System), so the Human Interface is mentioned occasionally in descriptions of individual commands. The commands exist as program files in the SYSTEM directory. When you type a command on the terminal, the Operating System looks for the file having that name in the default directory (\$), then in the PROG directory, then in the SYSTEM directory.

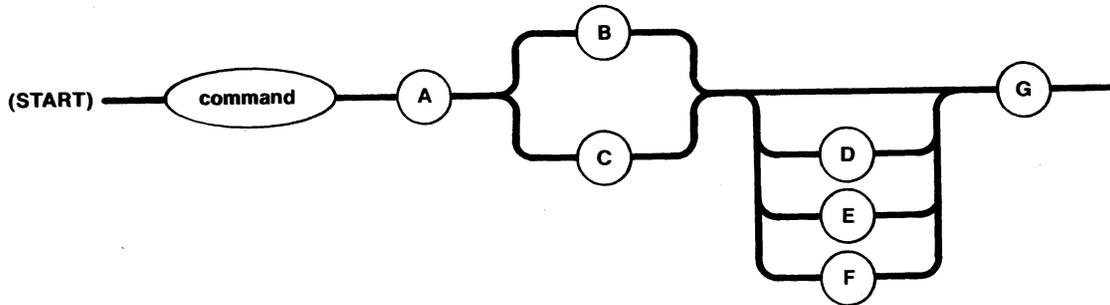
These commands are presented in alphabetical sequence without regard for functional organization. A functional grouping of the commands is given in the Human Interface Command Dictionary in Table 3-1 for fast reference.

COMMAND SYNTAX SCHEMATICS

The syntax for each command described in this chapter is presented by means of a "railroad track" schematic, with syntactic elements scattered along the track. Your entrance to any given schematic is always from left to right, beginning with some command name entry.

Elements shown in uppercase characters must be typed in a command line exactly as shown in the command schematics except that you can type them either in uppercase or lowercase characters. Elements shown in lowercase characters are generic terms, which means that you supply the specific item, such as the pathname for a file. The example that follows shows the possible paths through a railroad track schematic. Notice that the main track goes through required elements in a given command.

"Railroad sidings" go through optional parameter elements. In some cases, you have a choice of going through one of several possible sidings before returning to the main track. If the main track itself divides into two separate tracks, you select one parameter or the other but not both.



In this example:

- A is a required element.
- Either B or C is required but not both.
- D, E, or F are all optional but only one can be selected.
- G is required.

You can abbreviate command parameters instead of typing the entire parameter. To abbreviate a parameter, type as many characters as are required to make the parameter name unique. For example, the ATTACHDEVICE command has two parameters, NAMED, and PHYSICAL; you can abbreviate NAMED to N and PHYSICAL to P.

You cannot abbreviate either the command name or the prepositions (TO, OVER, AFTER, AS).

Table 3-1. iRMX™ 86 Command Dictionary

Command	Synopsis	Page
File and Volume Management Commands		
ATTACHDEVICE	Attaches a new physical device to the system and adds its logical name to the root job's object directory.	3-5
BACKUP	Copies named files to a backup volume.	3-8
COPY	Creates new data files, or copies files to other pathnames.	3-15
CREATEDIR	Creates one or more new directories.	3-18
DELETE	Deletes data files and empty directories from a volume on secondary storage.	3-22
DETACHDEVICE	Removes a physical device from system use and deletes its logical name from the root job's object directory.	3-24
DIR	Lists a directory's filenames (and optionally, file attributes).	3-25
DISKVERIFY	Verifies the data structures of named and physical volumes.	3-32
DOWNCOPY	Copies files and directories from an iRMX 86 volume mounted on a secondary storage device to an ISIS-II secondary storage device.	3-37
FORMAT	Formats an iRMX 86 volume.	3-40
RENAME	Renames files or directories.	3-45
RESTORE	Copies files from a backup volume to a named volume.	3-48
UPCOPY	Copies files and directories from an ISIS-II secondary storage device to an iRMX 86 volume mounted on a secondary storage device.	3-59

iRMX™ 86 COMMANDS

Table 3-1. iRMX™ 86 Command Dictionary (continued)

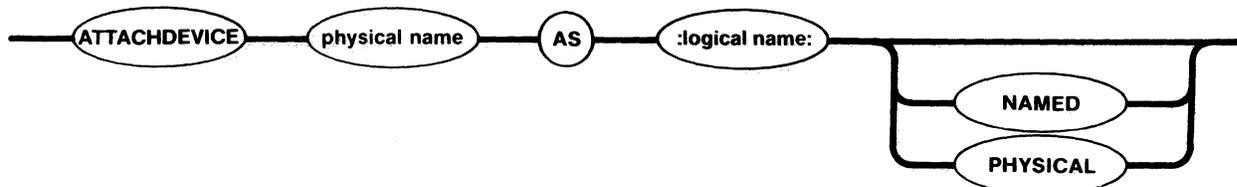
Command	Synopsis	Page
General Utility Commands		
DATE	Sets or resets the system date, or displays the current date.	3-20
DEBUG	Transfers control to the iSBC 957A/B package to debug an iRMX 86 application program.	3-21
SUBMIT	Reads, loads, and executes a string of commands from secondary storage instead of the keyboard.	3-55
TIME	Sets or resets the system clock, or displays the current system time.	3-58

IRMX™ 86 COMMANDS

ATTACHDEVICE

This command makes a physical device known to the system by a logical name. After the device is attached, it is accessed by commands and system calls with the logical name you specify.

The format of the command is as follows:



INPUT PARAMETERS

physical name	Physical name of the device to be attached to the system. These physical names were defined for your Operating System when the system was configured.
AS	Preposition; required for the command
:logical name:	This is the name that you assign to the device; it must be delimited with colons (:), and can be a maximum of 12 characters long including colons. After the device is attached with the ATTACHDEVICE command, any command or program code that accesses the device must specify the logical name.
NAMED	Specifies that the volume mounted on the device is already formatted for NAMED files. Examples of named-file volumes are diskettes or hard disk platters. If neither NAMED nor PHYSICAL are specified, NAMED is the default. See the FORMAT command in this chapter for a further description of NAMED files.
PHYSICAL	Specifies that the volume mounted on the logical device is already formatted as a single, large file. An example is a line printer. See the FORMAT command in this chapter for a further description of PHYSICAL volumes.

DESCRIPTION

A physical device must be attached to the system before it can be accessed; the device will be known by the logical name you assign. When you boot your iRMX 86 PC Operating System, the Operating System automatically attaches certain devices. These devices and logical names are listed in Chapter 1.

The most frequent use of the ATTACHDEVICE command is to attach a new device, such as a new disk drive or a line printer. For example, if you add a third disk drive to your system, you could attach it with the command

```
ATTACHDEVICE AFD2 AS :AFD2:
```

The logical name :AFD1: could just as well be :DISK3:, or any other name you wish to assign. The physical name AFD2 must be a name the Operating System has defined. The physical names for iRMX 86 PC disk drives are listed in a table in Chapter 5, PREPARING YOUR HARDWARE. The iRMX 86 PC Operating System has already assigned when the system is booted are listed in Chapter 1. (See the DETACHDEVICE command in this chapter for a description of how to detach a device from the system.)

When the attachment is completed, the ATTACHDEVICE command displays the following message:

```
physical name, attached as logical name
```

where "physical name" and "logical name" will be as specified in the ATTACHDEVICE command.

ERROR MESSAGES

```
logical name, device already attached
```

The specified physical device is already attached. ATTACHDEVICE does not attach the device.

```
device name, device does not exist
```

The physical device you specified is not a name the Operating System recognizes. ATTACHDEVICE does not attach the device.

```
logical name, invalid logical name
```

The logical name specification is not enclosed with colons, contains unmatched colons, is longer than 12 characters, or contains invalid characters. ATTACHDEVICE does not attach the device.

iRMX™ 86 COMMANDS

logical name, logical name already exists

The specified logical name is already used to attach a device. ATTACHDEVICE does not attach the device.

physical device, may not be attached as a (NAMED or PHYSICAL)

The NAMED or PHYSICAL specification in the command is not allowed for that physical device; for example, defining a line printer as a NAMED volume. ATTACHDEVICE does not attach the device.

008A : E\$CONTROL, too many device names

You tried to attach more than one physical device with a single ATTACHDEVICE command. ATTACHDEVICE does not attach a device.

logical name, volume is not a named volume

ATTACHDEVICE attempted to attach a device as a named device and discovered a physical volume on the device. However, ATTACHDEVICE does attach the device.

logical name, volume not formatted

ATTACHDEVICE attempted to attach a device as a named device and encountered an I/O error while searching for the volume's root directory. However, ATTACHDEVICE does attach the device.

logical name, volume not mounted

The specified device does not contain a volume; i.e., the diskette is not in the drive. However, ATTACHDEVICE does attach the device.

logical name, exception code

ATTACHDEVICE was unable to attach the specified device. This message lists the iRMX 86 exception code encountered. iRMX 86 exception codes are listed in Appendix A.

BACKUP

This command saves files in a named volume by copying them to a physical volume which serves as a backup volume. Later, you can use the RESTORE command (described later in this chapter) to retrieve these files and copy them to named volumes.

The format of this command is as follows:



INPUT PARAMETERS

pathname

Pathname of a file on the source volume. BACKUP saves files from the branch of the file tree that begins with the specified file. If you specify the logical name of the device only, BACKUP saves files beginning with the root directory of the volume.

'dd mmm yy'

Date parameter that BACKUP uses, in conjunction with the time parameter, to determine which files to save. BACKUP saves only those files that have been modified since the specified date and time. You must enclose the date parameter in single quotes. The individual fields of this parameter are:

dd Two-digit number that specifies the day of the month.

mmm Three-character abbreviation for the month, as follows:

JAN	APR	JUL	OCT
FEB	MAY	AUG	NOV
MAR	JUN	SEP	DEC

yy Two-digit number that specifies the year.

If you omit this parameter but specify the time parameter, the date defaults to the current system date. If you omit both the date and time parameters, the date defaults to 1 JAN 78.

INPUT PARAMETERS (continued)

hh:mm:ss Time parameter that BACKUP uses, in conjunction with the date parameter, to determine which files to save. BACKUP saves only those files that have been modified since the specified date and time. The individual fields of this parameter are:

hh Hours specified as 0-24.

mm Minutes specified as 0-59.

ss Seconds specified as 0-59.

If you omit this parameter, the time defaults to 00:00:00.

QUERY

Causes the Human Interface to prompt for permission to save each file. The Human Interface prompts with one of the following queries:

pathname, BACKUP data file?

or

pathname, BACKUP directory?

Enter one of the following responses to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Save the file.
E or e	Exit from the BACKUP command.
R or r	Continue saving files without further query.
Any other character	If data file, do not save the file; if directory file, do not save the directory or any file in that portion of the directory tree. Query for the next file, if any.

OUTPUT PARAMETER

:backup device: Logical name of the device to which BACKUP copies the files.

BACKUP

DESCRIPTION

BACKUP is a utility which saves named files on backup volumes, such as diskettes. BACKUP saves the following information for each file:

- File name
- Access list
- Extension data
- User ID of the file owner
- File granularity
- Contents of the file

You can copy this information back to a named file by using the RESTORE utility, described later in this chapter.

Before a volume can be used as a backup volume, the volume must be formatted. Although BACKUP will accept both physical and named volumes, it is recommended that you supply freshly-formatted physical volumes or old backup volumes for this purpose. BACKUP issues a message before continuing if the backup volume you supply is anything other than a freshly-formatted physical volume. When BACKUP copies files to the backup volume, it overwrites any information that currently exists on the volume.

In order for BACKUP save files from a named volume, you must have read access to the files and to the directories that contain them.

You can limit the files which BACKUP processes in the following ways:

- If you specify a complete directory name instead of just the device's logical name in the invocation line, BACKUP limits its processing to the specified directory and all subdirectories.
- If you specify the date and time parameters, BACKUP processes only those files modified since the specified time.
- If you specify the QUERY parameter, BACKUP asks permission before saving each file. If you deny permission for BACKUP to save a data file, BACKUP skips the file and continues with the next file. If you deny permission for BACKUP to save a directory file, BACKUP skips the directory and all files contained in the directory or its subdirectories.

When you enter the BACKUP command, BACKUP displays the following sign-on message:

```
iRMX 86 DISK BACKUP UTILITY, Vx.x
```

where Vx.x is the version number of the utility. It then prompts you for a backup volume.

DESCRIPTION (continued)

Whenever BACKUP requires a new backup volume, it displays the following message:

backup device, mount backup volume #nn, enter Y to continue:

where backup device indicates the logical name of the backup device and nn the number of the requested volume. (BACKUP in some cases displays additional information to indicate problems with the current volume.) In response to this message, place a volume in the backup device and enter one of the following:

<u>Entry</u>	<u>Action</u>
Y, y, R or r	Continue the backup process.
E or e	Exit from the BACKUP command.
Any other character	Invalid entry; reprompt for entry.

BACKUP continues prompting for a backup volume until you supply one that it can access.

If the backup volume you supply is not a freshly-formatted physical volume, but one that BACKUP can access (such as a named volume, a previously-used backup volume, or a physical volume containing data), BACKUP informs you of this with one of the following messages:

backup device, not a physical volume, enter Y to overwrite:

or

backup device, backup volume #nn, date, enter Y to overwrite:

where backup device is the logical name of the backup device, nn is the volume number of the backup volume, and date is the date on which the previous backup was performed. In response to these messages, enter one of the following:

<u>Entry</u>	<u>Action</u>
Y, y, R, or r	Use the volume as a backup volume, overwriting the information currently stored on the volume.
E or e	Exit from the BACKUP command.
Any other character	Reprompt for another volume.

BACKUP

DESCRIPTION (continued)

As BACKUP saves each file in the source volume, it displays the following message at the Human Interface console output device (:CO:):

pathname, SAVED

If your backup volume becomes full and you supply additional backup volumes, you should write the numbers of the backup volumes on the volume labels. Later, when you later restore files to a named volume with the RESTORE utility, you must supply the backup volumes in order.

ERROR MESSAGES

backup device, backup volume #nn, date, enter Y to overwrite:

The backup volume you supplied already contains backup information. BACKUP lists the logical name of the backup device, the volume number, and the date on which the original backup occurred. It overwrites this volume if you enter Y, y, R, or r.

backup device, cannot attach volume
backup device, exception code

backup device, mount backup volume #nn, enter Y to continue:

BACKUP cannot access the backup volume. This could be because there is no volume in the backup device, the volume is write protected, or because of a hardware problem with the device. The second line of the message indicates the iRMX 86 exception code encountered. BACKUP continues to issue this message until you supply a volume that BACKUP can access.

pathname, exception code, cannot back up file

For some reason BACKUP could not copy a file from the named volume, possibly because you do not have read access to the file or because there is a faulty area on the named volume. The message lists the pathname of the file and the exception code encountered. BACKUP copies as much of the file as possible and continues with the next file.

backup device, error writing volume label
backup device, exception code

backup device, mount backup volume #nn, enter Y to continue:

When BACKUP attempted to write a label on the backup volume, it encountered an error condition, possibly because of a faulty area on the volume, or because the volume is not formatted. The second line of the message indicates the iRMX 86 exception code encountered. BACKUP reprompts for a different backup volume.

ERROR MESSAGES (continued)

pathname, file does not exist

The pathname you specified as input to BACKUP does not represent an existing file or device.

backup device, invalid backup device

The logical name you specified for the backup device was not a logical name for a device.

exception code, invalid DATE or TIME

For either the DATE or TIME parameter, you entered a value that is out of range (such as 31 FEB 81 or 26:03:62). The message lists the exception code encountered as a result of this entry.

backup device, invalid logical name

The logical name you specified for the backup device contains unmatched colons, is longer than 12 characters, contains invalid characters, or does not exist.

backup device, not a physical volume, enter Y to overwrite:

The backup volume you supplied was formatted as a named volume or contained some other information. BACKUP will overwrite this volume if you enter Y, y, R, or r.

output specification missing

You did not supply the logical name of the backup device when you entered the BACKUP command.

keyword, too many values

You entered too many values with either the DATE or TIME parameters. The keyword portion of the message indicates the parameter that is in error.

keyword, unrecognized control

You entered one of the optional parameters of the form "keyword=value," but the keyword was not DATE, TIME, or QUERY.

BACKUP

ERROR MESSAGES (continued)

backup device, volume not formatted

backup device, mount backup volume #nn, enter Y to continue:

The backup volume you supplied was not formatted. BACKUP continues to issue this message until you supply a formatted backup volume.

backup device, write error on backup volume
backup device, exception code

BACKUP encountered an error condition when writing information to the backup volume. The second line of the message lists the exception code encountered. This error is probably the result of a faulty area on the volume.

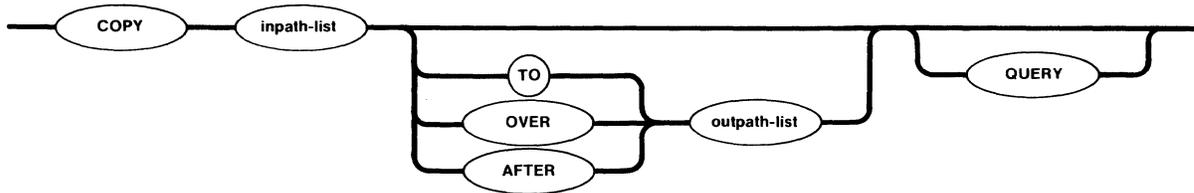
pathname, exception code

The pathname you specified as input to BACKUP is in error. This error could occur if you specify the same logical name that you specified for the backup device. It could also occur if you specify an invalid or nonexistent path component. This message displays the exception code that results from this error.

COPY

This command reads data from the specified input source or sources and writes the output to the specified destination file or files.

The format of the command is as follows:



INPUT PARAMETERS

inpath-list One or more pathnames for the files to be copied. Multiple pathnames must be separated by commas. Separating blanks are optional. To copy files on a one-for-one basis, you must specify the same number of files in the inpath-list as in the outpath-list.

QUERY Causes the Human Interface to prompt for permission to copy each file. Depending upon the specified preposition (TO, OVER, or AFTER), the Human Interface prompts with one of the following queries:

- pathname, copy TO out-pathname?
- pathname, copy OVER out-pathname?
- pathname, copy AFTER out-pathname?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Copy the file.
E or e	Exit from COPY command
R or r	Continue copying files without further query.
Any other character	Do not copy this file; go to the next file in the input list.

OUTPUT PARAMETERS

TO	Writes the listed input files to named new output files. The specified output file or files should not already exist; if they do, COPY will request permission to delete the existing files before it executes the copy operation for that file. If more input files than output files are listed, the remaining input files will be appended to the end of the last listed output file.
OVER	Writes the listed input files over (replaces) the existing output files on a one-for-one basis, regardless of file size. If an output file does not already exist, its corresponding input file is written to a new file with the listed output file name. If more input files than output files are listed, the remaining input files will be appended to the end of the last listed output file.
AFTER	Appends the input file or files to the current data in the existing output file or files. If the output file does not already exist, all listed input files will be concatenated into a new file with the listed output file name.
outpath-list	One or more pathnames for the output files. Multiple pathnames must be separated by commas. Separating blanks are optional. If the preposition and output parameter defaults are exercised in the command line, the output will go to the user's console screen (TO :CO:).

DESCRIPTION

COPY is a powerful and versatile command with a wide range of file handling applications (See Chapter 2 for examples). Implementation depends upon your selection of a preposition and your input file and output file specification in the command line. The following are some of the COPY command's features:

- Create new files (TO preposition).
- Copy over existing files or create new files (OVER preposition).
- Add data to the end of existing files (AFTER preposition).
- Copy a list of files to another list of files on a one-for-one basis.
- Concatenate two or more files into a single output file.

DESCRIPTION (continued)

As each file is copied, the COPY command displays one of the following messages, as appropriate:

```
pathname, copied TO out-pathname

pathname, copied OVER out-pathname

pathname, copied AFTER out-pathname
```

If you do not specify a preposition or output file, TO :CO: is the default output. The Human Interface normally expects all listed output files to be new files when the TO preposition is used; however, it is prepared to deal with existing files. If an existing output file name is encountered during a copy operation using TO, the Human Interface displays the the following message:

```
pathname, already exists, DELETE?
```

Enter Y or y if you wish to delete the existing file. The COPY command will delete the file.

Enter any other character if you do not wish to delete the existing file. The COPY command will pass over the corresponding input file without copying it, and will attempt to copy the next listed input file to its corresponding output file.

If more input files than output files are specified, the remainder of the input files will be appended to the end of the last listed output file. As each file is appended, the following message is displayed on the console screen:

```
pathname, copied AFTER out-file
```

If there are fewer input filenames than output filenames specified in the COPY command (regardless of the preposition), the output files remaining after the last valid copy operation will be ignored.

You cannot successfully use COPY to copy a directory to a data file or to another directory. Although a directory can be copied, the attributes of the directory are lost. That is, the directory can no longer be used as a directory. However, a file listed under one directory can be copied to another directory. For example:

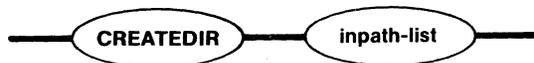
```
copy samp/test/a to :fl:/alpha/beta
```

would copy the A data file to a different volume, directory, and filename, where the new file's pathname would be :fl:/alpha/beta.

CREATEDIR

This command creates one or more iRMX 86 user directories.

The format is as follows:



INPUT PARAMETER

inpath-list	One or more pathnames of the iRMX 86 directories to be created. Multiple pathnames must be separated by commas. Embedded blanks between commas and pathnames are optional.
-------------	--

DESCRIPTION

A created iRMX 86 directory allows all access functions; that is, you can read/write, delete, list, add, and change the contents of the directory you created with CREATEDIR.

The following message is displayed if a directory is successfully created:

directory-name, directory created

You can create new directories that are subordinate to other directories. For example:

createdir ab/dc/ef/GH

would cause the newly-created directory GH to be nested within existing directory EF, which in turn, is nested within directory DC, and so on.

It is suggested that you use uppercase letters when you enter a new directory name in a CREATEDIR command, and use lowercase letters when you create a new data filename in a COPY command. You can then easily distinguish between directory names and filenames in a directory listing.

You can check the contents of the directory at any time by using the DIR command to list the directory (see the DIR command in this chapter).

ERROR MESSAGES

directory name, invalid file type

ERROR MESSAGES (continued)

You attempted to create a directory using a data file as part of the new directory's pathname; only other directory names are allowed in the pathname for a new directory.

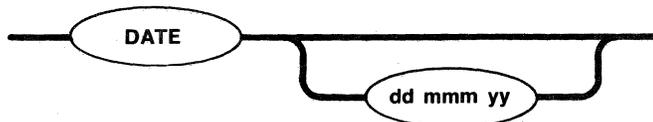
directory-name, directory already exists

The specified pathname of the directory to be created already exists.

DATE

This command sets a new system date or displays the current date.

The format is as follows:



INPUT PARAMETERS

dd	Two-digit number that specifies the day of the month.
mmm	Three-character abbreviation for the specified month, as follows:
	JAN APR JUL OCT
	FEB MAY AUG NOV
	MAR JUN SEP DEC
yy	Two-digit number that specifies the year.

DESCRIPTION

The dd, mmm, and yy entries are separated by single blanks.

If no new date is specified in the DATE command, the current date is displayed.

If one of the date entries in the parameter string is set, all three must be; there are no default settings for individual entries within the parameter string.

If you request the system date on a non-timing system, the following message will be displayed:

00:00:00

See also the TIME command in this chapter if you wish to set the system clock in conjunction with setting the date.

ERROR MESSAGES

Errors in a date entry, such as syntax errors or a number out of range (i.e., 31 FEB 81), cause the following error message to be displayed:

illegal date

If this occurs, reenter the DATE command with the correct syntax.

DEBUG

This command allows you to debug your iRMX 86 application jobs if your system is configured with the iSBC 957A/B package.



INPUT PARAMETERS

command pathname	Pathname of the file containing the application program to be debugged.
parameter string	String of required, optional, and default parameters that can be used in the command line to load and execute the application program.

DESCRIPTION

DEBUG loads your specified application program into main memory and transfers control to the iSBC 957A/B package. You can then use the iSBC 957A/B package to single-step, display registers, and set breakpoints within the program. Refer to the appropriate iSBC 957A or iSBC 957B user's guide for a complete description of the iSBC 957A/B functions.

When DEBUG executes, the 957A/B package runs with its interrupts disabled. Therefore, the time-keeping function is also disabled, with the following consequences:

- Impacts the ability of the Nucleus to execute time-out tasks that have provided time limits to system calls, such as RECEIVE\$UNITS and RECEIVE\$MESSAGE.
- Impacts the ability of the Basic I/O System to keep track of the time-of-day and write its data structures to secondary storage.

The 957A/B package cannot tolerate interrupts while the single-stepping command is being used. Single-stepping will be affected if:

- Tasks are using non-zero time-out values in system calls such as RECEIVE\$UNITS and RECEIVE\$MESSAGE.
- Time-of-day is configured in the Basic I/O System.
- Non-zero update timeout values are specified in the Basic I/O System's Device Unit Information Blocks (DUIB).

The alternative to single-stepping is to use breakpoints.

DELETE

This command removes data files and empty directories from secondary storage.

The format is as follows:

**INPUT PARAMETERS**

- inpath-list** One or more pathnames for the files or empty directories to be deleted. Multiple pathname entries must be separated by commas. Separating blanks are optional.
- QUERY** Causes the DELETE command to ask for your permission to delete each file in the list. Prior to deleting a file, the DELETE command displays the following query:

pathname, DELETE?

Enter one of the following, followed by a carriage return, in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Delete the file.
E or e	Exit from DELETE command.
R or r	Continue deleting without further query.
Any other character	Do not delete file; query for next file in sequence.

DESCRIPTION

The DELETE command allows you to release unused secondary storage space for new uses by removing empty directories and unneeded data files. If a file to be deleted is currently attached, it will be marked for deletion and deleted when the file is detached.

The following message is displayed as each file is deleted or marked for deletion:

pathname, deleted

ERROR MESSAGES

pathname, DELETE access required

You do not have permission to delete a specified file.

pathname, does not exist

The specified file was not found (e.g., a syntax error in a pathname or the file is located in some other directory). The DELETE command will attempt to delete each succeeding file specified in the filename-list after it has encountered an error in a file name.

pathname, directory not empty

Non-empty directories may not be deleted. You attempted to delete a directory that still lists filenames or other directory names.

If you still wish to delete the directory, you must first delete all its contents. For example, if you wished to delete a directory named ALPHA that contained a data file with the pathname ALPHA/BETA/SAMP, you would enter the following command:

```
delete alpha/beta/samp,alpha/beta,alpha
```

which would delete all files cataloged in ALPHA before the directory itself was deleted.

DETACHDEVICE

This command detaches the specified logical device.

DETACHDEVICE **:logical name:**

INPUT PARAMETER

:logical name: Logical name of the physical device that is to be deleted from the root job's object directory.

DESCRIPTION

The **DETACHDEVICE** command allows you to detach a device without having to reconfigure the system. After a device is detached, no volume mounted on that device is accessible for system use. For a description of formatted volumes (**NAMED** or **PHYSICAL**), see the **FORMAT** command description in this chapter.

When the device is detached and its logical name has been deleted from the root job's object directory, the **DETACHDEVICE** command will display the following message:

logical-name, detached

NOTE

Using the **DETACHDEVICE** command to detach the device containing your Human Interface commands causes loss of access to Human Interface functions until the system is restarted.

ERROR MESSAGE

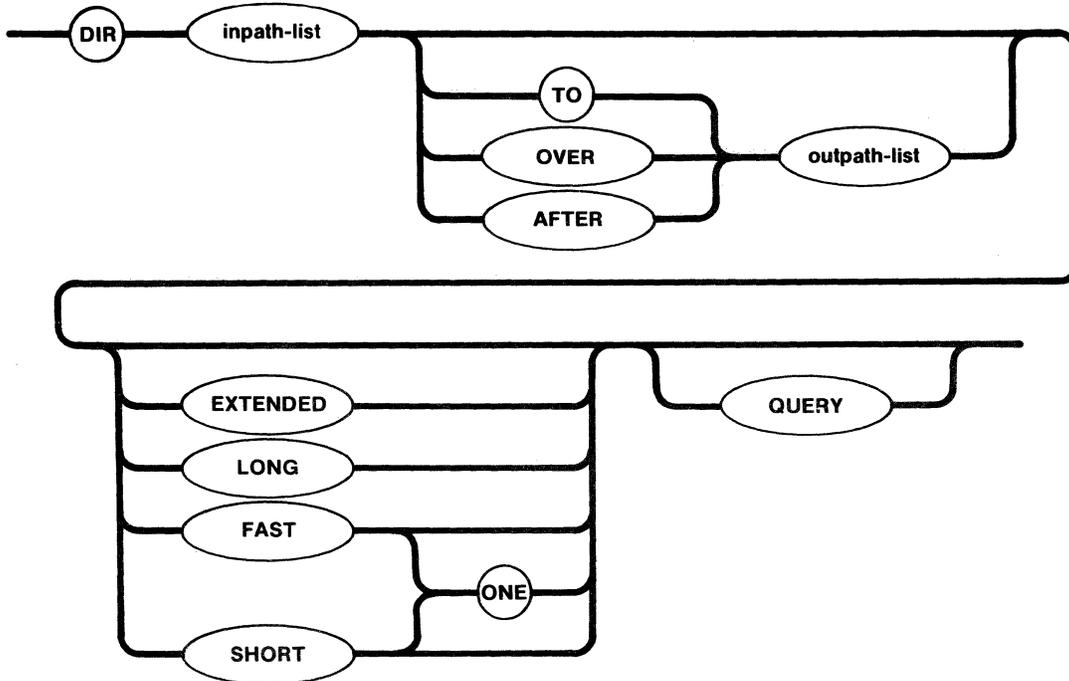
illegal logical name

Either there is a syntax error in the logical name specification or the logical name does not exist in the root job's object directory.

DIR

This command lists the names and attributes of files contained in a given directory, including data filenames and directory names.

The format of the command is as follows:



INPUT PARAMETERS

inpath-list One or more pathnames of the directories to be listed. Multiple directory pathname entries must be separated by commas. Separating blanks are optional. If no pathname is specified, the user's default directory is listed.

EXTENDED Lists all available information for each data file or directory file in the directory. The first line for each file will be the same as for the LONG form. The second line will contain the last access date, creation date, and the accessor list. The listing will be in a double-column format (see Figure 3-1 at the end of this command description).

LONG Lists file information in a one-line format (see Figure 3-2 at the end of this command description).

INPUT PARAMETERS (continued)

- FAST** Lists only the filenames and directory names in the directory. The output format will be five columns of filenames unless you also specify the ONE parameter (see Figure 3-3 at the end of this command description). If no listing format is specified, FAST is the default.
- SHORT** Lists the file information in a two-column format (see Figure 3-4 at the end of this command description).
- ONE** Lists the output of a FAST or SHORT listing in single-column format. ONE is the default number of columns for EXTENDED or LONG listings.
- QUERY** Causes the DIR command to prompt you for permission to list a directory by issuing the following message:

pathname, DIR?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	List the directory.
E or e	Exit from DIR command.
R or r	Continue listing directories without further query.
Any other character	Do not list directory; query for the next directory, if any.

OUTPUT PARAMETERS

- TO** Copies the directory listing to the specified destination data file. If no TO/OVER/AFTER preposition is specified, TO :CO: is the default.
- OVER** Copies the directory listing to the specified output file and writes over (replaces) the previous contents.
- AFTER** Appends the directory listing to the current contents of the specified output file.
- out-pathname** Pathname of the file to receive the directory listing. If the parameter is omitted, the default destination is the user's console screen (TO :CO:).

DESCRIPTION

The amount of information listed for each file depends upon what listing format you specify (EXTENDED, FAST, LONG, or SHORT) in the DIR command. An example of each type of listing format is provided at the end of the DIR command description in Figures 3-1 through 3-4 respectively. An explanation of the illustrated headings is provided in Table 3-2 following the figures.

If you want to list the default user directory but also wish to specify a listing format other than FAST, use the default directory name explicitly. For example:

```
dir $ extended
```

would display a listing of the user directory in the EXTENDED format. Note that the default directory is a configuration option.

ERROR MESSAGES

```
pathname, is not a directory
```

A pathname exists but is not a directory.

```
pathname, directory does not exist
```

The pathname does not exist, either as a directory or as a data file.

```
pathname, directory LIST access required
```

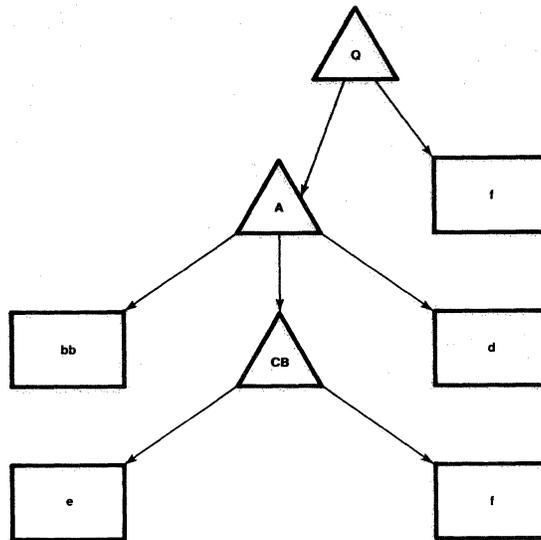
You do not have list access to the directory.

DIR COMMAND EXAMPLES

The examples that follow show how a directory's files are listed when you use your configured system's default prefix in a directory's pathname. In the examples, directory names are enclosed in triangles; data file names are enclosed in rectangles.

Assume you have the following directory structure for your files:

DIR COMMAND EXAMPLES (continued)



If your root directory was Q, then the following files would be listed in response to the DIR pathname entry examples in the following "Pathname" column:

<u>Pathname</u>	<u>Files Listed</u>
omitted	A, f
f	not allowed because f is a data file
A	bb, CB, d
A/d	not allowed because d is a data file
A/CB	e, f
A/CB/e	not allowed because e is a data file

DIR LISTING FORMATS

Figures 3-1, 3-2, 3-3, and 3-4 show output examples for EXTENDED, FAST, LONG, and SHORT listing formats respectively. Table 3-2 defines the displayed column headings.

```

11 MAR 80 06:30:30
  DIRECTORY OF sys ON myvol

          GRAN
NAME      AT  ACC  BLKS  LENGTH  VOL FIL OWNER      LAST MOD
ed                DR    200    30185   16  3 Beck      20 NOV 79
          CREATION: 20 APR 78  ACCESSORS  ACC
          LAST ACC: 25 NOV 79  Engineers  R
                                Techs        U

idisk     DR  DLAC    5      39    16  1 WORLD    12 DEC 79
          CREATION: 15 NOV 78  ACCESSORS  ACC
          LAST ACC: 10 JAN 80

submit$plm$ab MA  DRAU    11    1057   24  2 BACKWORDPLMCOM 16 JUN 79
          CREATION: 20 APR 78  ACCESSORS  ACC
          LAST ACC: 20 JAN 80  PYE-WACKET AU
                                TOGAN        R
                                longlongname D  U

CHAOTICGOOD          U 123456789 1234567890 12345 123 Chopin    01 DEC 79
          CREATION: 16 NOV 79  ACCESSORS  ACC
          LAST ACC: 20 FEB 80  Clerics    RA
                                MAGIC-USERS  U
                                Thieves      D
                                FIGHTERS     DRAU

LAWFULEVIL          D          73    9081   24  15 saveyourhat  04 JAN 80
          CREATION: 15 NOV 79  ACCESSORS  ACC
          LAST ACC: 05 MAR 80  WORLD      RAU

          5 FILES      1839 BLOCKS  1200453 BYTES

```

Figure 3-1. EXTENDED Directory Listing Example

```

11 MAR 80 04:25:40
  DIRECTORY OF alpha ON mvol
fname1 fname2 fname3 fname4 fname5
fname6 fname7 fname8 fname9 fname10
fname11 . . .
.
.
.

```

Figure 3-2. FAST Directory Listing Example

DIR LISTING FORMATS (continued)

```

11 JAN 80 06:30:30
  DIRECTORY OF sys ON myvol

                                GRAN
NAME      AT  ACC      BLKS    LENGTH  VOL FIL OWNER      LAST MOD
ed        DR  DR        200    30185   16  3 BECK             20 NOV 79
idisk     DR  DLAC      5      39      16  1 WORLD            12 DEC 79
LEMONADEIT MA  D        105    13074   64  2 malagi           16 MAR 77
credit    DR  DR        263    32967  128  6 WORLD            17 NOV 79
SUBMITAGAINPLM MA DRAU     11    1057   24  2 BACKWORDPLMCOM 16 JUN 79
type      DR  LA        4      366    16  1 PASCAL            15 DEC 79
CHAOTICGOOD      U 123456789 1234567890 12345 123 CHOPIN            01 DEC 79
LAWFULEVIL      D        73     9081   24  15 saveyourpet     04 JAN 80
  8 FILES      1839 BLOCKS    1200453 BYTES

```

Figure 3-3. LONG Directory Listing Example

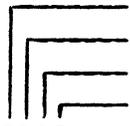
```

11 MAR 80 04:25:50
  DIRECTORY OF sys ON myvol
NAME      AT  ACC  BLKS  LENGTH  NAME      AT  ACC  BLKS  LENGTH
append    R    R    40    1425   attrib    DRAU 38   4682
COPY      MA  DRAU 65    8042   CREDIT.HAZ R    263  33017
dcopy     DRAU 62    7718   DELETE    A    37   4506
REFERENCE DR  L    5     10     DATA    DR  DLAC 1    4
DUMP      D    D    22    2568   ED        DR   200  30185
idisk     DR  DLAC 5     39
LEMONADEIT MA  D    123456789 1234567890
CREDIT    DR  DRAU 263   32967   RENAME    AU   21   2487
submit$plm MA  DRAU 11    057    TYPE     DR  LA   4    366
CHAOTICGOOD      U 13293 1151640 lawfulevil D    73   9081
  18 FILES      1839 BLOCKS    1200453 BYTES

```

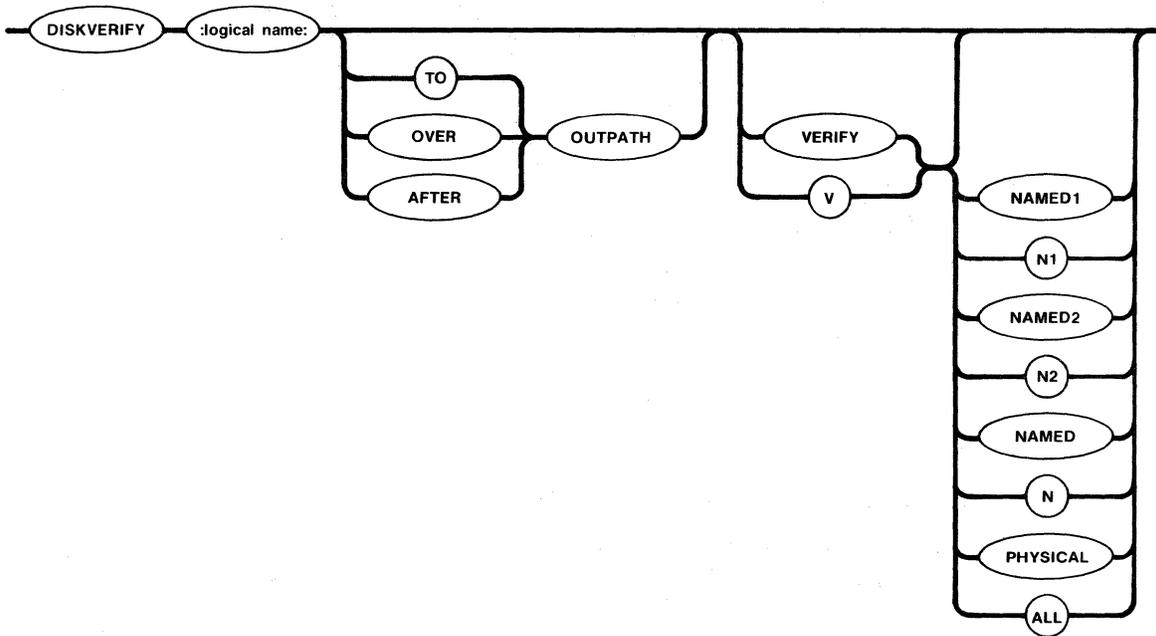
Figure 3-4. SHORT Directory Listing Example

Table 3-2. Directory Listing Headings

Heading	Meaning
NAME:	14-character file NAME
AT:	File ATtribute, where: DR = Directory (if the ATtribute field is blank, the file is a data file)
ACC:	File ACCess rights, where: <div style="display: flex; align-items: center; margin-left: 40px;"> <div style="margin-right: 10px;">Directories:</div> <div style="margin-right: 10px;">DLAC</div> <div style="margin-right: 10px;">  </div> <div> <ul style="list-style-type: none"> Delete List Add Change </div> </div> <div style="display: flex; align-items: center; margin-left: 40px;"> <div style="margin-right: 10px;">Other Files:</div> <div style="margin-right: 10px;">DRAU</div> <div style="margin-right: 10px;">  </div> <div> <ul style="list-style-type: none"> Update Append Read Delete </div> </div>
BLKS:	Nine-digit number (five digits on SHORT listing) giving the volume-granularity units allocated to the file. On the SHORT form, if the number of digits exceeds five, the file is displayed in the nine-digit form (see the LEMONADEIT file in Figure 3-4).
LENGTH:	10-digit number (7 digits on SHORT listing) giving the length of the file in bytes. On the SHORT form, if the number of digits exceeds 7, the file is displayed in the 10-digit form (see the LEMONADIT file in Figure 3-4).
VOL:	Five-digit number giving the volume granularity in bytes.
FIL:	Three-digit number giving the granularity of the file in volume-granularity units.
OWNER:	14-character, alphanumeric owner name.
LAST MOD:	Date of last file modification.
LAST ACC:	Date of last file access.
CREATION:	Date of file creation.
ACCESSORS:	Heading for list of 14-character accessor names.
ACC:	Heading for access rights of file accessors. The format of this field is identical to ACC above.

DISKVERIFY

This command invokes the disk verification utility which verifies the data structures of iRMX 86 physical and named volumes. This utility can also be used to reconstruct portions of the volume and perform absolute editing on the volume. The format of the DISKVERIFY command is as follows:



INPUT PARAMETERS

- :logical-name:** Logical name of the secondary storage device containing the volume.
- VERIFY or V** Performs a verification of the volume. If you specify this parameter and omit the options, the utility performs the NAMED verification.
- If you specify this parameter, the utility performs the verification function and returns control to you at the Human Interface level. You can then enter any Human Interface command.
- If you omit this parameter, the utility displays a sign-on message and the utility prompt (*). You can then enter individual disk verification commands. These commands are described in the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL.

INPUT PARAMETERS (continued)

NAMED1 or N1	VERIFY option that applies to named volumes only. This option checks the fnodes of the volume to ensure that they match the directories in terms of file type and file heirarchy. This option also checks the information in each fnode to ensure that it is consistent. As a result of this option, DISKVERIFY displays a list of all files on the volume that are in error, with information about each file. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
NAMED2 or N2	VERIFY option that applies to named volumes only. This option checks the allocation of fnodes on the volume, checks the allocation of space on the volume, and verifies that the fnodes point to the correct locations on the volume. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
NAMED or N	VERIFY option that performs both the NAMED1 and NAMED2 verification functions on a named volume. If you omit the VERIFY option, NAMED is the default option.
PHYSICAL	VERIFY option that applies to both named and physical volumes. This option reads all blocks on the volume and checks for I/O errors.
ALL	VERIFY option that applies to both named and physical volumes. For named volumes, this option performs both the NAMED and PHYSICAL verification functions. For physical volumes, this option performs the PHYSICAL verification function.

OUTPUT PARAMETERS

TO	Copies the output from the disk verification utility to the specified file. If no preposition is specified, TO :CO: is the default.
OVER	Copies the output from the disk verification utility over the specified file.
AFTER	Appends the output from the disk verification utility to the end of the specified file.

DISKVERIFY

OUTPUT PARAMETERS (continued)

outpath Pathname of the file to receive the output from the disk verification utility. If you omit this parameter and the TO/OVER/AFTER preposition, the utility copies the output to the console screen (TO :CO:). You cannot direct the output to a file on the volume being verified. If you attempt this, the utility returns an E\$NOT_CONNECTED error message.

DESCRIPTION

When you enter the DISKVERIFY command, the utility responds by displaying the following line:

```
iRMX 86 DISK VERIFY UTILITY, Vx.x
```

where Vx.x is the version number of the utility. If you specify the VERIFY or V parameter in the DISKVERIFY command, the utility performs a verification of the volume and copies the verification information to the console (or to the file specified by the outpath parameter). Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for a description of the verification output. After generating the verification output, the utility returns control to the Human Interface, which prompts you for more Human Interface commands. The following is an example of such a DISKVERIFY command:

```
-DISKVERIFY :F1: VERIFY NAMED2
iRMX 86 DISK VERIFY UTILITY , Vx.x
DEVICE NAME = F1          : DEVICE SIZE = 0003E900 : BLOCK SIZE = 0080

'NAMED2' VERIFICATION

      BIT MAPS O.K.
```

However, if you omit the VERIFY (or V) parameter from the DISKVERIFY command, the utility does not return control to the Human Interface. Instead, it issues an asterisk (*) as a prompt and waits for you to enter individual DISKVERIFY commands. The following is an example of such a DISKVERIFY command:

```
-DISKVERIFY :F1:
*
```

After you receive the asterisk prompt, you can enter any of the DISKVERIFY commands listed in the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL.

ERROR MESSAGES

logical name, 0045 : E\$LOG_NAME_NEXIST

You specified a nonexistent logical name in either the :logical name: parameter or the outpath parameter.

8042 : E\$NOT_CONNECTION

You attempted to direct output to a file on the volume being verified.

command line error

You made a syntax error when entering the command.

device size inconsistent

size in volume label = value1 : computed size = value2

When the disk verification utility computed the size of the volume, the size it computed did not match the information recorded in the iRMX 86 volume label. It is likely that the volume label contains invalid or corrupted information. This error is not a fatal error, but it is an indication that further error conditions may result during the verification session. You may have to reformat the volume or use the disk verification utility to modify the volume label. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about the disk verification utility commands.

logical name, illegal logical name

The logical name you specified was not surrounded by colons (:).

not a named disk

You tried to perform a NAMED, NAMED1, or NAMED2 verification on a physical volume.

verify-function argument error

The VERIFY option you specified is not valid.

The NAMED1, NAMED2, and PHYSICAL verification options can also produce error messages. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about these messages.

DISKVERIFY

EXAMPLE

The following command performs both named and physical verification of a named volume.

-DISKVERIFY :F1: VERIFY ALL

DEVICE NAME = F1 : DEVICE SIZE = 0003E900 : BLK SIZE = 0080

'NAMED1' VERIFICATION

'NAMED2' VERIFICATION

BIT MAPS O.K.

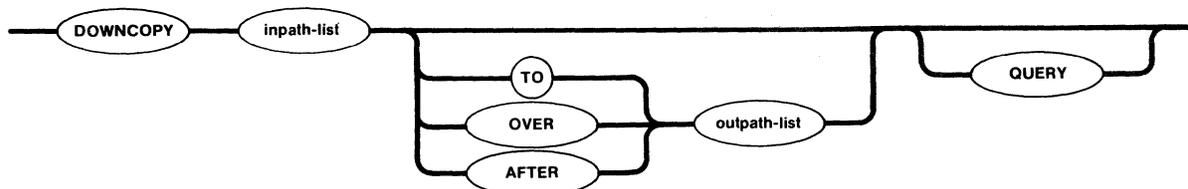
'PHYSICAL' VERIFICATION

NO ERRORS

-

DOWNCOPY

This command copies files from a volume on an iRMX 86 secondary storage device to a volume on an ISIS-II secondary storage device via the iSBC 957A/B Interface and Execution package. The format is as follows:



INPUT PARAMETERS

inpath-list One or more iRMX 86 pathnames for files, separated by commas, that are to be copied to ISIS-II secondary storage. Separating blanks between pathnames are optional. The files may be copied in the listed sequence either on a one-for-one basis or concatenated into one or more files.

QUERY Causes the Human Interface to prompt for permission to copy each iRMX 86 file to the listed ISIS-II destination file. Depending on which preposition you specify (TO, OVER, or AFTER), the Human Interface prompts with one of the following queries:

pathname, copy down TO out-pathname?

pathname, copy down OVER out-pathname?

pathname, copy down AFTER out-pathname?

Enter one of the following in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Copy the file.
E or e	Exit from the DOWNCOPY command.
R or r	Continue copying files without further query.
Any other character	Do not copy this file; query for the next file in sequence.

DOWNCOPY

OUTPUT PARAMETERS

- TO** Reads iRMX 86 files and copies them TO new ISIS-II files in the listed sequence. The specified output files should not already exist in the ISIS-II directory when the TO parameter is used. If a named output file does exist, DOWNCOPY will display the following message:
- filename, already exists, delete?
- Enter a Y or y if you wish to delete the existing file. Enter any other character if you do not wish the existing file to be deleted.
- If no preposition is specified, TO :CO: (ISIS-II console screen) is the default. If more input files than output files are specified, the remaining input files will be appended to the end of the last listed ISIS-II file.
- OVER** Reads the listed iRMX 86 input files and copies them OVER the existing ISIS-II destination files in the listed sequence. If more input files than output files are listed, the remaining input files will be appended to the end of the last listed ISIS-II file.
- AFTER** Reads the listed iRMX 86 input files and copies them AFTER the end of data on the existing ISIS-II destination files in the listed sequence.
- outfile-list** One or more ISIS-II filenames for the output files. Multiple filenames must be separated by commas. Separating blanks are optional. If the preposition and output file defaults are used in the command line, the output will go to the ISIS-II console screen.

DESCRIPTION

The DOWNCOPY command cannot be used to copy directories from an iRMX 86 system to an ISIS-II system; only files can be copied.

Before you enter a DOWNCOPY command on the iRMX 86 console keyboard, you must have your target system connected to a development system with the 957A/B package, and the package must be running. The ISIS-II copies of the files will have all ISIS-II file attributes turned off.

As each file in the input list is copied, one of the following messages will be displayed on the Human Interface console output device (:CO:), as appropriate:

DESCRIPTION (continued)

pathname, copied down TO out-filename

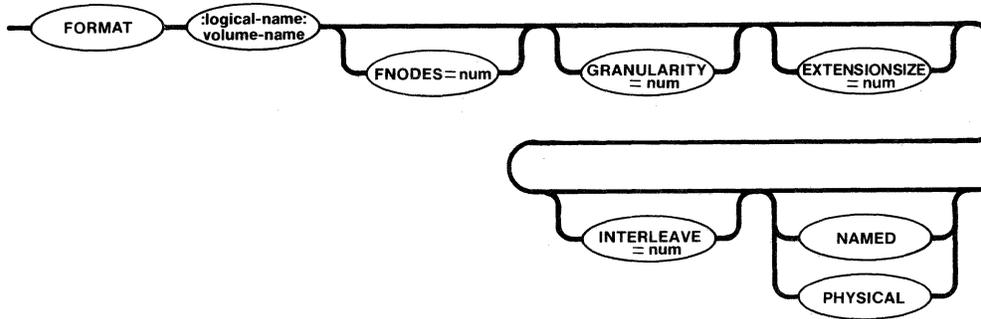
pathname, copied down OVER out-filename

pathname, copied down AFTER out-filename

FORMAT

This command formats or reformats a volume on an iRMX 86 secondary storage device, such as a diskette, hard disk, or bubble memory.

The format is as follows:



INPUT PARAMETERS

- :logical-name:** Logical name of the physical device-unit to be formatted. The logical name must be preceded and followed by colons without embedded blanks between the logical name and volume name.
- volume-name** Six-character, alphanumeric ASCII name, without embedded blanks, to be assigned to the volume. (See the definition for a "volume" in Chapter 1.)
- FNODES=num** Defines the maximum decimal number of files that may be created on a NAMED volume. (This parameter is not meaningful when formatting a PHYSICAL volume and will be ignored if specified for such volumes.) The range is 7 through 32,767 fnodes, although the maximum number of fnodes you can define depends on the settings of the GRANULARITY and EXTENSIONSIZE parameters (as explained in the "Description" portion of this command write-up). If not specified, the default is 50 fnodes.
- GRANULARITY=num** Volume granularity; the minimum number of bytes to be allocated for each increment of file size on a NAMED volume. (This parameter is not meaningful for PHYSICAL volumes, and will be ignored if specified for such volumes.) The specified decimal number is placed in the header of the volume and becomes the default file granularity when a file is created on the volume.

INPUT PARAMETERS

GRANULARITY=num (continued)

The range is 1 through 65,535 (decimal) bytes, although the maximum allowable volume granularity depends on the settings of the FNODES and EXTENSIONSIZE parameters (as explained in the "Description" portion of this write-up). If not specified, the default granularity is the device granularity. Once the volume granularity is defined, it applies to every file created on that volume.

EXTENSIONSIZE=num Size, in bytes, of the extension data portion of each file descriptor node (fnode). (This parameter is not meaningful for PHYSICAL volumes, and will be ignored if specified for such volumes.) The range is 0 through 65,448 (decimal), although the maximum allowable extension size depends on the settings of the FNODES and GRANULARITY parameters (as explained in the "Description" portion of this write-up). If not specified, the default extension size is 3 bytes.

INTERLEAVE=num Interleave factor for a NAMED or PHYSICAL volume. If not specified, the default value is 5, which is the optimum interleave factor for an iSBC 204 bootstrap load. See the interleave discussion under "Description" in this command write-up.

NAMED The volume can store only named files; that is, the volume can hold many files (up to the number of fnodes allocated), each of which can be accessed by its pathname. A diskette or hard disk surface are examples of devices that would be formatted for named files. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the type of files specified when you attached the device (with the ATTACHDEVICE command).

PHYSICAL The volume can be used only as a single, physical file. The GRANULARITY and FNODES parameters are not meaningful when PHYSICAL is specified for the volume. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the type of files specified when you attached the device (with the ATTACHDEVICE command).

DESCRIPTION

Every physical device-unit used for secondary storage must be formatted before it can be used for storing and then accessing its files. For example, every time you mount a previously unused diskette into a drive, you must enter a FORMAT command to format that diskette as a new volume before you can create, store and access files on it.

Once a volume is formatted, its name becomes a volume identifier when you list the root directory for the volume, and the name will appear in the directory's heading. Although the Human Interface uses the volume name in its own internal processing when you access the volume, you do not need to specify the volume name in any subsequent command after the volume is formatted; only the logical name of the secondary storage device on which the volume is currently mounted needs to be specified.

The number of fnodes on a volume defines the number of files that can exist on the volume. You can specify this number with with the FNODES parameter. Each fnode is a data structure that contains information about a file. Each time you create a file on the volume, the Operating System records information about the file in an unused fnode. Later, it uses the fnode in order to determine the location of the file on the volume.

Each fnode contains a field that stores extension data for its associated file. An operating system extension can access and modify this extension data by invoking the A\$GET\$EXTENSION\$DATA and A\$SET\$EXTENSION\$DATA system calls (refer to the iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL for more information). When you format a volume, you can use the EXTENSIONSIZE parameter to set the size of the extension data field in each fnode. Although you can specify any size from 0 to 65,448 bytes, the Human Interface requires all fnodes to have at least 2 bytes of extension data.

The default volume granularity is always the granularity of the physical device for the volume. For example, if the default granularity for a device is 128 bytes of secondary storage, the I/O System will automatically allocate 128 bytes of permanent storage to each new file you create on that volume, regardless of whether or not a file requires a full 128 bytes. If the size of a file exceeds 128 bytes, the I/O System will allocate still another full block of 128 bytes, and so on, until the volume is full.

Although the FNODES, GRANULARITY, and EXTENSIONSIZE parameters have maximum values which are listed in the parameter descriptions, the combination of these three parameters must also satisfy the following formula:

$$(87 + \text{EXTENSIONSIZE}) \times \text{FNODES} / \text{GRANULARITY} \leq 65535$$

where all numbers are decimal. FORMAT displays an error message if the combination of parameter values exceeds the limit.

DESCRIPTION (continued)

As stated previously, the interleave factor applies to volumes formatted either for NAMED or PHYSICAL files. The interleave specification maximizes access speed for the files on a given volume, depending on the intent of volume and the device configuration. For example, an interleave factor of 5 for a flexible disk drive means that, for each file, the I/O System will read every fifth sector on the diskette, starting with an index of 1 (other, hard disk systems may be different, depending on your configuration). Therefore, the I/O System does not need to wait for the disk to make a complete revolution before it accesses the next sector; the next sector by an increment of 5 is ready to be accessed for read/ write by the time the previously accessed sector has been processed.

The FORMAT command displays the following message when volume formatting is completed:

```

volume (vol-name) will be formatted as a NAMED/PHYSICAL volume
granularity = number
interleave = number
numberfnodes = number
extensionsize = number
    
```

where:

vol-name	The volume name specified in the FORMAT command.
NAMED/PHYSICAL	Either NAMED or PHYSICAL will be displayed in the message, depending on the command specification.
number	Default or specifically defined in the command.

ERROR MESSAGES

If a device cannot be detached for formatting, the following message is displayed on the user console:

```

logical-name, can't detach device
    
```

which means that the volume does not exist, the volume is busy, or the device on which the volume is mounted is not currently attached to the system.

If the device cannot be attached for formatting, or it cannot be re-attached (e.g., restored to its original configuration prior to formatting) after formatting takes place, the following message is displayed on the user console:

```

device-name, can't attach device
    
```

ERROR MESSAGES (continued)

The following error message is displayed if you attempt to format something that is not a physical device:

logical-name, is not a device connection

The following error message is displayed if you specify a volume name containing more than six ASCII characters or if you specify a logical device name:

vol-name, illegal name

The following error message is displayed if you specify an out-of-range number for any of the FNODES, GRANULARITY, EXTENSIONSIZE, or INTERLEAVE parameters:

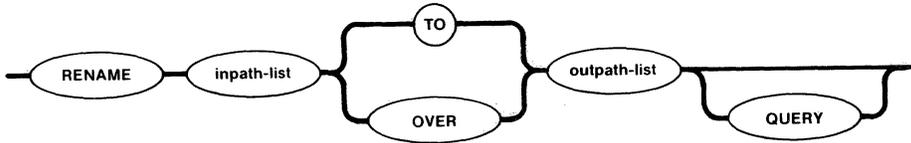
number, illegal number

The following message is displayed if the values you specify for fnode size, granularity, and extension data size cause the formula listed in the "Description" section to exceed its limit.

vol-name, fnode file size exceeds 65535 volume blocks

RENAME

This command allows you to change the pathname of one or more data files or directories. RENAME is effective across directory boundaries on the same volume. The format is as follows:



INPUT PARAMETERS

inpath-list One or more pathnames, separated by commas, of files or directories that are to be renamed. Blanks between pathnames are optional separators.

QUERY Causes the Human Interface to prompt for permission to rename each pathname in the input list by issuing the following message:

oldname, RENAME?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Rename the file.
E or e	Exit from RENAME command.
R or r	Continue renaming without further query.
Any other character	Do not rename file; query for the next file in sequence.

OUTPUT PARAMETERS

TO Moves the data to the new pathnames in the output list. A new pathname in the output list should not already exist. If, in fact, a new pathname does already exist, RENAME displays the following warning message when the pre-existing pathname is encountered:

pathname, already exists, DELETE?

OUTPUT PARAMETERS

- TO (continued)** Enter a "Y" or "y" if you wish the pre-existing pathname and its contents to be written over by the new name specification. The pre-existing pathname and its contents will be deleted.
- Enter any other character if you do not wish the pre-existing file to be deleted. Renaming of the specified file will not take place and the **RENAME** command will attempt to rename the next pathname in the list sequence.
- OVER** Changes each old pathname in a list to the corresponding new pathname, even if the new pathname already exists. The old pathname is deleted from secondary storage. **OVER** cannot be used to rename a non-empty directory over another non-empty directory.
- outpath-list** List of new pathnames. Multiple pathnames must be separated by commas. Separating blanks are optional.

DESCRIPTION

The primary distinction between the **RENAME** command and the **COPY** command is that, as a **RENAME** command is executed, it releases the pathnames in the listed input files for new uses without having to perform any further operation on the files.

Although **RENAME** can be used to rename an existing directory pathname **TO** a new pathname, it cannot be used to rename an existing directory **OVER** another existing directory. For example:

```
-rename ALPHA to DELTA          allowed
-rename ALPHA over BETA         not allowed (unless BETA is empty)
-rename ALPHA/sampl over BETA/testl    allowed
```

CAUTION

Note that changing the name of a directory also changes the path of all files listed under that directory. All subsequent accesses to those files must specify the new pathnames for the files.

As each file in a pathname list is renamed, the **RENAME** command displays one of the following messages, as appropriate:

```
old-pathname, renamed TO new-pathname
      or
old-pathname, renamed OVER new-pathname
```

ERROR MESSAGES

There must be a one-for-one correspondence between the oldname and newname lists in the RENAME command. A missing element in either list causes RENAME to display the following message:

unmatched path name lists

If your system is configured with user-designed access limitations, you must have at least delete access to old pathnames and add-entry access to the destination directory to use the RENAME command.

If you are not allowed delete access on your system, the following message is displayed when you attempt to use the OVER preposition in a RENAME command:

old-pathname, DELETE access required

If you are not allowed add-entry access on your system, the following message is displayed when you attempt to use the TO preposition in a RENAME command:

new-pathname, directory ADD ENTRY access required

If the RENAME command encounters an error in the renaming of a file, it will attempt to continue renaming each succeeding file in sequence.

Use of the AFTER preposition is not valid for the RENAME command, and an attempt to use it causes the following message to be displayed:

AFTER preposition, TO or OVER preposition expected

Note that the RENAME command is the only Human Interface file handling command that cannot be used across volume boundaries; that is, you cannot use the RENAME command to rename a file or move data from a volume located on one secondary storage device to a volume located on another secondary storage device (e.g., from one diskette to another). An attempt to do so causes the following error message:

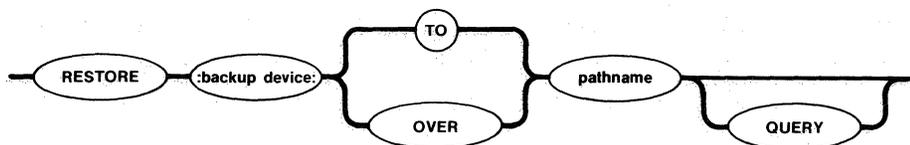
0005: E\$CONTEXT

Use the COPY command or a combination of COPY and DELETE commands if you wish to rename files or move data across volume boundaries.

RESTORE

This command restores files to a named volume by copying them from a backup volume.

The format of this command is as follows:

**INPUT PARAMETERS**

:backup device: Logical name of the backup device from which RESTORE restores files.

QUERY Causes the Human Interface to prompt for permission to restore each file. The Human Interface prompts with one of the following queries:

pathname, RESTORE data file?

or

pathname, RESTORE directory?

Enter one of the following responses to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Restore the file.
E or e	Exit from the RESTORE command.
R or r	Continue restoring files without further query.
Any other character	If data file, do not restore the file; if directory file, do not restore the directory or any file in that portion of the directory tree. Query for the next file, if any.

OUTPUT PARAMETERS

TO	Restores the files from the backup volume to new files on the named volume, if the files do not already exist on the named volume. However, if a file being restored already exists on the named volume, RESTORE prompts for permission to restore the file.
OVER	Restores the files from the backup volume over (replaces) the files on the named volume. If a file does not exist on the named volume, RESTORE creates a new file on the named volume.
pathname	Pathname of a file which receives the restored files (you must specify a directory pathname when restoring more than one file). If you specify a logical name for a device, RESTORE places the files under the root directory for that device. However, the device must contain a volume formatted as a named volume. If you wish to restore files to the directory in which they originated, you should specify the same pathname parameter as you used with the BACKUP command.

DESCRIPTION

RESTORE is a utility which copies files from backup volumes (where the BACKUP command originally saved them) to named volumes. RESTORE copies the files to any directory you specify, maintaining the hierarchical relationships between the backed-up files.

When RESTORE copies files, it copies only those files for which you are the owner. For these files, it restores the following information:

- File name
- Access list
- Extension data
- File granularity
- Contents of the file

RESTORE changes the creation, last modification, and last access dates of the file to the current date.

Each backup volume which is used as input to the RESTORE command must contain files placed there by the BACKUP command. In addition, if the backup operation required multiple backup volumes, you must restore these volumes in the same order as they were backed up.

RESTORE

DESCRIPTION (continued)

The output volume which receives the restored files must be a named volume. You must have sufficient access rights to the files in that volume to allow RESTORE to perform all necessary operations. In order for RESTORE to create new files on a named volume, you must have add entry access to directories on that volume. In order for RESTORE to restore files over existing files, you must have add entry and change entry access to directories in that volume and delete, append, and update access to data files.

When you enter the RESTORE command, RESTORE displays the following sign-on message:

```
IRMX 86 DISK RESTORE UTILITY Vx.x
```

where Vx.x is the version number of the utility. Then it prompts you for a backup volume.

Whenever RESTORE requires a new backup volume, it issues the following message:

```
backup device, mount backup volume #nn, enter Y to continue:
```

where backup device indicates the logical name of the backup device and nn the number of the requested volume. (RESTORE in some cases displays additional information to indicate problems with the current volume.) In response to this message, place the backup volume in the backup device (make sure that the volume number is correct if the backup operation involved multiple volumes). Enter one of the following:

<u>Entry</u>	<u>Action</u>
Y, y, R, or r	Continue the restore process.
E or e	Exit from the RESTORE command.
Any other character	Invalid entry; reprompt for entry.

RESTORE continues prompting you until you supply the correct backup volume.

As it restores each file, RESTORE displays the following message at the Human Interface console output device (:CO:):

```
pathname, RESTORED
```

However, if a file with the same pathname already exists during a restore operation using the TO preposition, RESTORE displays the following message:

```
pathname, already exists, DELETE?
```

DESCRIPTION (continued)

Enter one of the following in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Delete the file and replace it with the one from the backup volume.
E or e	Exit from the RESTORE command.
R or r	Delete the file, replace it with the one from the backup volume, and continue restoring files without further queries.
Any other character	Do not restore the file; go on to the next file.

ERROR MESSAGES

pathname, ADD ENTRY or UPDATE access required

RESTORE could not restore a file, either because you did not have add entry access to the file's parent directory or because you did not have update access to the file. RESTORE continues with the next file.

backup device, backup volume #nn, date, mounted
 backup device, backup volume #nn, date, required

backup device, mount backup volume #nn, enter Y to continue:

RESTORE cannot continue because the backup volume you supplied is not the one that RESTORE expected. Either you supplied a volume out of order or you supplied a volume from a different backup session. RESTORE reprompts for the correct backup volume.

backup device, cannot attach volume
 backup device, exception code

backup device, mount backup volume #nn, enter Y to continue:

RESTORE cannot access the backup volume. This could be because there is no volume in the backup device, the volume is write protected, or because of a hardware problem with the device. The second line of the message indicates the iRMX 86 exception code encountered. RESTORE continues to issue this message until you supply a volume that RESTORE can access.

ERROR MESSAGES (continued)

pathname, DELETE access required

RESTORE could not restore a file because you did not have delete access to the file. RESTORE continues with the next file.

pathname, exception code, error during BACKUP, file not restored

When the BACKUP utility saved files, it encountered an error when attempting to save the file indicated by this pathname. RESTORE is unable to restore this file. The message lists the iRMX 86 exception code encountered.

pathname, exception code, error during BACKUP, restore incomplete

When the BACKUP utility saved the files, it encountered an error when attempting to save the file indicated by this pathname. RESTORE restores as much of the file as possible to the named volume. The message lists the iRMX 86 exception code encountered.

backup device, error reading backup volume
backup device, exception code

RESTORE tried to read the backup volume but encountered an error condition, possibly because of a faulty area on the volume. The second line of the message indicates the iRMX 86 exception code encountered.

pathname, exception code, error writing output file, restore incomplete

RESTORE encountered an error while writing a file to the named volume. This message lists the iRMX 86 exception code encountered. RESTORE writes as much of the file as possible to the named volume.

pathname, extension data not completely restored, nn bytes required

The amount of space available on the named volume for extension data is not sufficient to contain all the extension data associated with the specified file. The value nn indicates the number of bytes required to contain all the extension data. This message indicates that the named volume on which RESTORE is restoring files is formatted differently than the named volume which originally contained the files. RESTORE restores as much of the extension data as possible. To ensure that you restore all the extension data from the backup volume, you should restore the files to a volume formatted with an extension size set equal to the largest value reported in any message of this kind. Refer to the description of the FORMAT command for information about setting the extension size.

ERROR MESSAGES (continued)

pathname, file does not exist

The pathname you specified as input to RESTORE does not represent an existing file or device.

pathname, file not restored

For some reason, RESTORE was unable to restore a file from the backup volume. RESTORE continues with the next file. Another message usually precedes this message to indicate the reason for not restoring the file.

backup device, invalid logical name

The logical name you specified for the backup device contains unmatched colons, is longer than 12 characters, contains invalid characters, or does not exist.

backup device, not a backup volume

backup device, mount backup volume #nn, enter Y to continue:

The volume you supplied on the backup device was not a backup volume. RESTORE continues to issue this message until you supply a backup volume.

backup device, not a valid backup device

The logical name you specified for the backup device was not a logical name for a device.

output specification missing

You did not specify a pathname to indicate the destination of the restored files.

pathname, READ access required

You do not have read access to a file on the backup volume; therefore RESTORE cannot restore the file.

keyword, too many values

You specified too many values after the TO or OVER parameter.

RESTORE

ERROR MESSAGES (continued)

keyword, unrecognized control

You entered an invalid optional parameter. The keyword portion of the message indicates the parameter that is in error.

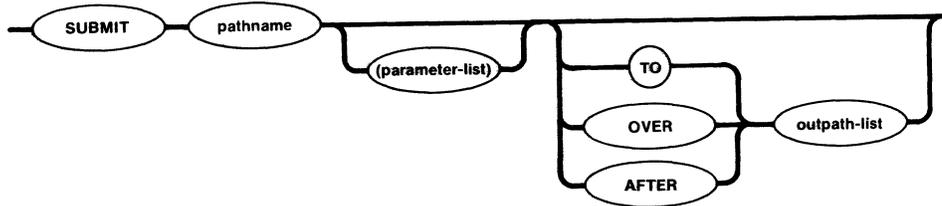
pathname, exception code

The pathname you specified as input to RESTORE is in error. This error could occur if you specify an invalid or nonexistent path component. This message displays the exception code that results from this error.

SUBMIT

This command reads and executes a set of commands from a file in secondary storage instead of from the console keyboard. To use the SUBMIT command you must first create a data file that defines the command sequence and formal parameters (if any).

The format of the command is as follows:



INPUT PARAMETERS

pathname	Name of the file from which the commands will be read. This file may contain nested SUBMIT files.
parameter-list	Actual parameters that are to replace the formal parameters in the SUBMIT file. You must surround this parameter list with parentheses. You can specify as many as 10 parameters, separated by commas, in the SUBMIT command. If you omit a parameter, you must reserve its position by entering a comma. If a parameter contains a comma, space, or parenthesis, you must enclose the parameter in single quotes. The sum of all characters in the parameter list must not exceed 512 characters.

OUTPUT PARAMETERS

TO	Causes the output from each command in the SUBMIT file to be written to the specified new file instead of the console screen. If the listed output file already exists, the SUBMIT command will display the following message:
----	--

pathname, already exists DELETE?

Enter a Y or y if you wish the existing output file to be deleted. Enter any other character if you do not wish the existing file to be deleted. A response other than Y or y causes the SUBMIT command to be terminated and you will be prompted for a new command entry.

OUTPUT PARAMETERS (continued)

OVER	Causes the output for each command in the SUBMIT file to be written over the specified existing file instead of the console screen.
AFTER	Causes the output from each command in the SUBMIT file to be written to the end of an existing file instead of the console screen.
outpath-list	Pathnames of one or more files to receive the processed output from each command executed from the SUBMIT file. If no preposition or output file is specified, TO :CO: is the default.

DESCRIPTION

Any program that reads its commands from the console keyboard can be executed from a SUBMIT file. If another SUBMIT command is itself used in a SUBMIT file, it causes another SUBMIT file to be invoked. You can nest SUBMIT files to any level of nesting until memory is exhausted. When one nested SUBMIT file completes execution, it returns control to the next higher level of SUBMIT file.

When you create a SUBMIT file, you indicate formal parameters by specifying the characters %n, where n ranges from 0 through 9. When SUBMIT executes the file, it replaces the formal parameters with the actual parameters listed in the SUBMIT command (the first parameter replaces all instances of %0, the second parameter replaces all instances of %1, and so forth). If the actual parameter is surrounded by quotes, SUBMIT removes the quotes before performing the substitution. If there is no actual parameter that corresponds to a formal parameter, SUBMIT replaces the formal parameter with a null string.

When you specify a preposition and output file in a SUBMIT command, only your SUBMIT command entry will be echoed on the console screen; the individual command entries in the submit file are not displayed on the screen as they are loaded and executed.

The SUBMIT command will display the following message when all commands in the submit file have been executed:

END SUBMIT pathname

EXAMPLE

This example shows a SUBMIT file that uses formal parameters and the command that you can enter to invoke this SUBMIT file. The SUBMIT file, which resides on file :F1:PROGRAM, contains the following lines:

```
ATTACHDEVICE F1 AS %0
CREATEDIR %0/%1
UPCOPY :F1:%2 TO %0%1/%2
```

The SUBMIT file contains three formal parameters, indicated by %0, %1, and %2. The %0 indicates the logical name of an iRMX 86 device; the %1 indicates the name of a directory on that device; the %2 indicates the name of a file which will be copied from an ISIS-II disk to the iRMX 86 device.

The SUBMIT command used to invoke this file is as follows:

```
-SUBMIT :F1:PROGRAM (:F1:, PROG, FILE1)
```

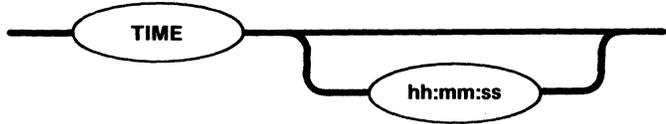
The command sequence created and executed by SUBMIT is shown as it would be echoed on the console output device.

```
-ATTACHDEVICE F1 AS :F1:
F1, attached as :F1:
-CREATEDIR :F1:/PROG
:F1:PROG, directory created
-UPCOPY :F1:FILE1 TO :F1:PROG/FILE1
:F1:FILE1 upcopied TO :F1:PROG/FILE1
END SUBMIT :F1:PROGRAM
-
```

TIME

This command sets the system clock. If no new time is entered, the TIME command causes the current system time to be displayed.

The format is as follows:



INPUT PARAMETERS

- hh: Hours specified as 0 through 24.
- mm: Minutes specified as 0 through 59.
- ss Seconds specified as 0 through 59.

DESCRIPTION

If one of the time entries in the parameter string is set, all three must be; there are no default settings for individual items in the parameter string.

If you request the time-of-day and the system clock has not been set, the TIME command displays the following message:

00:00:00

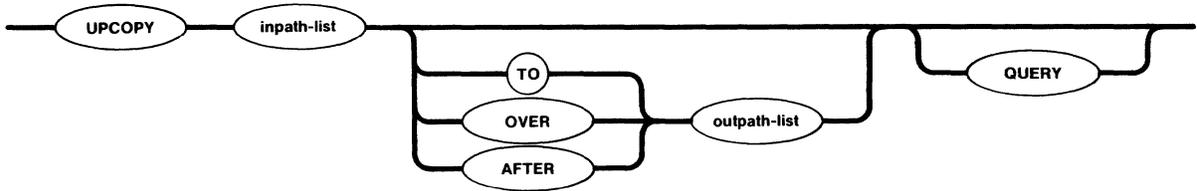
See also the DATE command in this chapter if you wish to set the date in conjunction with the system clock.

An invalid time or an out-of-range entry for the TIME command causes the following error message to be displayed:

illegal time

UPCOPY

This command copies files from a volume on ISIS-II secondary storage to a volume on iRMX 86 secondary storage via the iSBC 957A/B Interface and Execution package.



INPUT PARAMETERS

inpath-list List of one or more filenames of the ISIS-II files that are to be copied to iRMX 86 secondary storage, either on a one-for-one basis or concatenated into one or more iRMX 86 output files.

QUERY Causes the Human Interface to prompt for permission to copy each ISIS-II file to the listed iRMX 86 output file. Depending on which preposition you specify (TO, OVER, or AFTER), the Human Interface prompts with one of the following queries:

filename, copy up TO out-pathname?

filename, copy up OVER out-pathname?

filename, copy up AFTER out-pathname?

Enter one of the following (followed by a carriage return) in response to the query:

<u>Entry</u>	<u>Action</u>
Y or y	Copy the file.
E or e	Exit from the UPCOPY command.
R or r	Continue copying files without further query.
Any other character	Do not copy this file; go to the next file in sequence.

OUTPUT PARAMETERS

- TO** Copies the ISIS-II file or files TO a new iRMX 86 file or files in the listed sequence. The output file or files should not already exist when the TO preposition is used. If no preposition is specified, TO :CO: is the default. If more input files than output files are specified in the command line, the remaining inputfiles will be appended to the end of the last listed output file.
- OVER** Copies the listed ISIS-II input file or files OVER existing iRMX 86 destination files in the listed sequence. If more input files than output files are listed in the command line, the remaining input files will be appended to the end of the last listed output file.
- AFTER** Appends the listed ISIS-II input file or files AFTER the end-of-data on an existing iRMX 86 output file or files in the listed sequence.
- outpath-list** One or more pathnames of the iRMX 86 destination files. Multiple pathnames must be separated by commas. Separating blanks are optional. If the preposition and output parameter defaults are used in the command line, the output will go to the iRMX 86 console screen.

DESCRIPTION

Before you enter an UPCOPY command on the iRMX 86 console keyboard, you must have your target system connected to a development system with the 957A/B package and the package must be running. The iRMX 86 copies of the files will have WORLD access; that is, all iRMX 86 system users can perform read, write, and delete operations on the files without restriction.

As each ISIS-II file in the input list is copied, the Human Interface will display one of the following messages on the iRMX 86 console screen, as appropriate:

filename, copied up TO out-pathname

filename, copied up OVER out-pathname

filename, copied up AFTER out-pathname

CHAPTER 4. UDI SYSTEM CALLS

Your programs request iRMX 86 PC Operating System services through the Universal Development Interface (UDI) system calls. This chapter describes the set of system calls that are available to iRMX 86 PC programs. Although the iRMX 86 Operating System can recognize many other system calls, (these are listed in Appendix B) you can perform all normal operations with UDI calls. This is a design characteristic of the UDI; it forms a "membrane" through which your programs send requests to the Operating System, and through which the Operating System returns information to programs.

This chapter contains these four sections:

- USING THE UDI. This section outlines general programming considerations for using the Universal Development Interface. For example, this section explains how to use UDI libraries and how to deal with errors in programs.
- TYPES OF UDI SYSTEM CALLS. This section explains certain concepts about UDI File Management and Memory Management system calls. For example, the concept of a file connection is explained here.
- DESCRIPTIONS OF SYSTEM CALLS. Here is the heart of this chapter. Each UDI system call is described in detail, with an explanation of how the call is invoked. The calls are arranged alphabetically for quick reference. At the beginning of this section you will find a System Call Dictionary: a brief listing of system calls arranged into these functional groupings:
 - Memory Management
 - File Handling
 - Program Control
 - Exception Handling
 - Utility and Command Parsing
- EXAMPLE PROGRAM: At the end of this chapter is the listing of a PL/M-86 program that uses a representative sample of UDI system calls.

UDI SYSTEM CALLS

USING THE UDI

This section contains information about:

- UDI Libraries and INCLUDE files
- Exceptional conditions such as hardware errors
- Special data types referred to in descriptions of UDI system calls

UDI LIBRARIES

To execute a program which uses UDI system calls, you must link the program to one of three iRMX 86 UDI libraries. These libraries are called URXLRG.LIB, URXSML.LIB, and URXCOM.LIB. If your program corresponds to the LARGE or MEDIUM models of segmentation, link it to URXLRG.LIB. If your program corresponds to the SMALL or COMPACT models of segmentation, link it to URXSML.LIB or URXCOM.LIB, respectively. These libraries are in the UDI directory under the directory SYSTEM. The pathname for the COMPACT library, for example, is SYSTEM/UDI/URXCOM.LIB.

The iRMX 86 PROGRAMMING TECHNIQUES manual discusses selecting a model of segmentation. While these models deal with the PL/M 86 language, they apply to assembly language as well. In contrast, Pascal-86 and FORTRAN-86 require the large library.

INCLUDE FILES

You must declare each UDI procedure used in your PL/M-86 programs as an EXTERNAL procedure. These declarations are contained in a single file named SYSTEM/UDI/UDI.EXT. You INCLUDE this file with a PL/M-86 program that makes UDI system calls. You can edit this file to delete references that you don't use in your programs.

EXCEPTIONAL CONDITIONS

Every UDI call except DQ\$EXIT returns a condition code which specifies the status of the call. Each condition code has a unique numeric value, and an associated mnemonic by which it is known. For example, the code indicating that there were no errors or unusual conditions has the numeric value zero (0) and the name E\$OK. Any code other than E\$OK returned from a system call means there was an exceptional condition. Exception codes are classified as:

- Environmental Exceptions. These are generally caused by conditions outside the control of a program; for example, device errors or insufficient memory.

UDI SYSTEM CALLS

- Programmer Errors. These are typically caused by coding errors (for example, "bad parameter"), but "divide-by-zero", "overflow", "range check", and errors detected by the 8087 Numeric Processor Extension are also classified as avoidable.

When an error is detected, the normal (default) system action is to display on the console terminal an error message, and terminate the program. However, you may establish your own routine to handle exceptions by using the UDI system calls DQ\$TRAP\$EXCEPTION and DQ\$DECODE\$EXCEPTION.

Appendix A contains a list of exception codes that the iRMX 86 Operating System can return, with the numeric value, mnemonic, and meaning of each code.

DATA TYPES

The following data types are referred to in the descriptions of system calls:

BYTE	An 8-bit item.
WORD	A two-byte item.
STRING	A sequence of bytes, the first of which contains the length (in bytes) of the remaining portion of the string. A length of zero indicates a null string.
TOKEN	A WORD passed between a program and the Operating System to represent an object; for example, a CONNECTION is a token used in File Management System calls to represent a file. In PL/M-86: DECLARE TOKEN LITERALLY 'WORD';
POINTER	Equivalent to PL/M-86 type POINTER. It is two bytes under the small model of segmentation; four bytes in other cases.
CONNECTION	A token used to manipulate iRMX 86 files. In PL/M-86: DECLARE CONNECTION LITERALLY 'WORD';
SELECTOR	Equivalent to the PL/M-86 type SELECTOR; a 16-bit iAPX 86,88 paragraph number (the base portion of a four-byte pointer).

UDI SYSTEM CALLS

DESCRIPTIONS OF SYSTEM CALLS

This section contains descriptions of each UDI system call. The calls are arranged alphabetically. Before the first system call description, a System Call Dictionary (Table 4-1) shows the calls arranged in functional groups, with a short description of each call and the page number of the description.

Every system call description contains the following information in the order listed here:

- The name of the system call.
- A brief summary of the function of the call.
- The form of the call as it is invoked from a PL/M-86 program, with symbolic names for each parameter. (Calling sequences show formal parameters in lower case.)
- Definition of input and output parameters.
- A complete explanation of the system call, including any information you will need to use the system call.

NOTE

The first system call described, DQ\$ALLOCATE, also includes an actual (as opposed to formal) PL/M-86 invocation of the system call and an ASM-86 calling sequence. These are included only once because they are typical of all system calls.

MEMORY MANAGEMENT SYSTEM CALLS

When the iRMX 86 Operating System loads and runs a program, the program is allocated a specific amount of memory. The portion of memory not occupied by loaded code and data -- the free space pool -- is available to programs dynamically, i.e., while the program is running. The Operating System manages memory as segments of the size a program requests.

Your programs can use the UDI system calls DQ\$ALLOCATE, and DQ\$FREE, respectively, to get a memory segment from the pool, and to return the segment to the pool. You can use the call DQ\$GET\$SIZE to receive information about an allocated memory segment.

UDI SYSTEM CALLS

FILE-HANDLING SYSTEM CALLS

About one-half of UDI system calls are used to manipulate files. Figure 4-1 shows the chronological relationship between the most frequently used file-handling system calls.

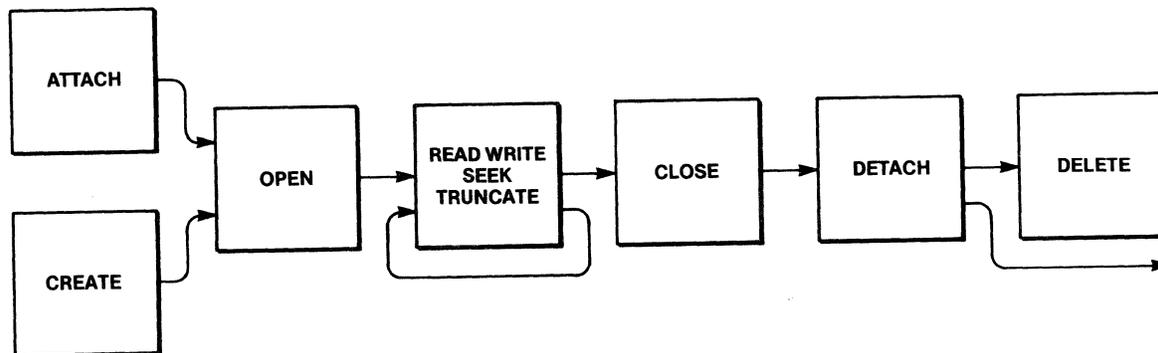


Figure 4-1. Chronology Of System Calls

The iRMX 86 Operating System distinguishes between:

- Establishing the association between a program and a data file
- Operating on the data file

The association between a program and a data file is a connection, and is represented in your programs by a token of type CONNECTION.

Your programs establish a connection by using the system calls DQ\$ATTACH or DQ\$CREATE and break the connection with DQ\$DETACH. When your program establishes a connection via DQ\$ATTACH or DQ\$CREATE, it receives a CONNECTION token from the operating system. You use this token in all further communications with the operating system to identify the file.

You use the procedure DQ\$OPEN to prepare an established connection for input/output operations. You perform the actual input or output operations with DQ\$READ and DQ\$WRITE. You can move the file pointer with the DQ\$SEEK call. When input/output is finished, you close the file connection with DQ\$CLOSE. Note that you open and close connections, not files. Closing a file connection frees buffer space. Once a connection is established, it may be opened and closed as often as necessary.

DQ\$DETACH is the call that eliminates a connection, and DQ\$DELETE deletes a file. If a file has connections attached when a program issues DQ\$DELETE, the Operating System will mark for deletion the file. That is, the file is not actually deleted until all connections are detached.

UDI SYSTEM CALLS

EXCEPTION-HANDLING SYSTEM CALLS

When an exceptional condition occurs while the iRMX 86 Operating System is running a user program, the default exception handler (part of the Operating System) will terminate the program and display a message on the terminal identifying the exception code. You can write a program to handle exception codes, rather than using the default exception handler. In this case, the Operating System will not terminate your program, but will pass control to your exception handler. Three system calls are used to define and use your own exception handler:

- DQ\$TRAP\$EXCEPTION, which is used to identify an exception handler that you provide.
- DQ\$GET\$EXCEPTION\$HANDLER, which is an informative system call returning the address of the current exception handler: either the default system handler, or one you specify with DQ\$TRAP\$EXCEPTION
- DQ\$DECODE\$EXCEPTION, which converts an exception numeric code into its equivalent mnemonic.

Before your exception handler gets control, the iRMX 86 Operating System does the following:

1. Pushes the condition code onto the stack.
2. Pushes the number of the parameter that caused the exception onto the stack (1 for the first parameter, 2 for the second, etc.).
3. Pushes a word onto the stack (reserved for future use).
4. Pushes a word for the 8087 Numeric Processor Extension onto the stack.
5. Initiates a long call to the exception handler.

If the condition was not caused by an erroneous parameter, the responsible parameter number is zero. If the exception code is E\$NDP, the fourth item pushed onto the stack is the 8087 status word, and 8087 exceptions have been cleared.

Programs compiled under the SMALL model of segmentation cannot have an alternate exception handler, but must use the default system exception handler. This is because the exception handler must have a LONG POINTER, which is not available with SMALL segmentation.

UDI SYSTEM CALLS

Table 4-1. SYSTEM CALL DICTIONARY

SYSTEM CALL	FUNCTION PERFORMED	PAGE
MEMORY MANAGEMENT CALLS		
DQ\$ALLOCATE	Creates a segment of a specified size for use by the application.	4-9
DQ\$FREE	Returns the specified segment to the system.	4-11
DQ\$GET\$SIZE	Returns the size of the specified segment.	4-25
FILE-HANDLING CALLS		
DQ\$ATTACH	Creates a connection to a specified file.	4-11
DQ\$CHANGE\$EX- TENSION	Changes the extension of a file name.	4-12
DQ\$CLOSE	Closes the specified file connection.	4-13
DQ\$CREATE	Creates a file for use by the application.	4-14
DQ\$DELETE	Deletes a file.	4-16
DQ\$DETACH	Closes a file and deletes its connection.	4-17
DQ\$GET\$CON- NECTION\$STATUS	Returns status of a file connection.	4-22
DQ\$OPEN	Opens a file for a particular type of access.	4-28
DQ\$READ	Reads the next sequence of bytes from a file.	4-32
DQ\$RENAME	Renames the specified file.	4-34
DQ\$SEEK	Moves the current position pointer of a file.	4-35
DQ\$SPECIAL	Sets terminal line-edit/tranparent mode.	4-37
DQ\$TRUNCATE	Truncates a file to the specified length.	4-41
DQ\$WRITE	Writes a sequence of bytes to a file.	4-42

UDI SYSTEM CALLS

Table 4-1. SYSTEM CALL DICTIONARY (continued)

SYSTEM CALL	FUNCTION PERFORMED	PAGE
PROGRAM CONTROL		
DQ\$EXIT	Exits from the current application job.	4-18
DQ\$OVERLAY	Causes the specified overlay to be loaded.	4-30
EXCEPTION-HANDLING CALLS		
DQ\$DE- CODE\$EXCEPTION	Returns a short description of a specified error code.	4-15
DQ\$GET\$EXCEPT- ION\$HANDLER	Returns a POINTER to the address of the program currently being used to process errors.	4-24
DQ\$TRAP\$EXCEPTION	Identifies a custom exception processing program for a particular type of error.	4-40
UTILITY AND COMMAND PARSING		
DQ\$GET\$ARGUMENT	Returns the next argument from the character string used to invoke the application program.	4-20
DQ\$GET\$SYS- TEM\$ID	Returns the name of the underlying operating system supporting the UDI.	4-26
DQ\$GET\$TIME	Returns the current time of day as kept by the underlying operating system.	4-27
DQ\$SWITCH\$BUFFER	Selects a new buffer from which to process commands.	4-39

DQ\$ALLOCATE requests a memory segment from the free memory pool.

```
base$addr = DQ$ALLOCATE (size, except$ptr);
```

INPUT PARAMETER

size	A WORD which, <ul style="list-style-type: none">• if not zero, contains the size, in bytes, of the requested segment. If the size parameter is not a multiple of 16, it will be rounded up to the nearest multiple of 16.• if zero, indicates that the size of the request is 65536 (64K) bytes.
------	---

OUTPUT PARAMETERS

base\$addr	A SELECTOR in which the Operating System places the base address of the memory segment. If the request fails because the memory requested is not available, this argument will be OFFFFH, and the system will return an E\$MEM exception code.
except\$ptr	A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

The DQ\$ALLOCATE system call is used to request additional memory. You may use this call for dynamically creating buffer space.

EXAMPLE CALL PROCEDURES

These examples are included only for DQ\$ALLOCATE. Their form is typical of all system calls.

EXAMPLE CALL PROCEDURES (continued)

Both examples request 128 (decimal) bytes of memory, and point to a word named "ERR" for receiving the condition code).

Example PL/M-86 Calling Sequence

```

DECLARE  ARRAY_BASE  WORD,
         ERR          WORD;
.
.
.
ARRAYBASE = DQ$ALLOCATE (128, @ERR);

```

Example ASM86 Calling Sequence

```

MOV     AX,128
PUSH   AX      ; first parameter
LEA    AX,ERR
PUSH   DS      ; second parameter
PUSH   AX      ;
CALL   DQ$ALLOCATE
MOV    ARRAYBASE,AX ; returned value

```

This example is applicable to programs assembled according to the COMPACT, MEDIUM, and LARGE models of segmentation. For the SMALL model, you would not push the segment register before each parameter.

The DQ\$ATTACH system call creates a connection to an existing file.

```
connection = DQ$ATTACH (path$ptr, except$ptr);
```

INPUT PARAMETER

path\$ptr A POINTER to a STRING containing the pathname for the file to be attached.

OUTPUT PARAMETERS

connection A WORD in which the iRMX 86 Operating System will place the CONNECTION to the file.

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call allows a program to obtain a connection to any file. Attaching a file that is already attached is valid. A connection to the existing file is made, and all prior connections remain established.

It is not a valid operation to attach :CO: or :LP:; if you do so the Operating System will return the exception code E\$SUPPORT.

DQ\$CHANGE\$EXTENSION changes or adds the extension at the end of a file name.

```
CALL DQ$CHANGE$EXTENSION (path$ptr, extension$ptr, except$ptr);
```

INPUT PARAMETERS

path\$ptr A POINTER to a STRING that specifies the path for the file to be renamed.

extension\$ptr A POINTER to a series of three bytes containing the characters that are to be added to the pathname. This is not a STRING. You must include three bytes, even if some are blank.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call is used to change a file name extension, or add an extension. For example: :AFD1:FILE.SRC can be changed to :AFD1:FILE.OBJ by a compiler when the compiler creates a file in which the object file is written.

The three character extension may not contain delimiters recognized by DQ\$GET\$ARGUMENT but may contain trailing blanks. If the first character addressed by extension\$ptr is a space, the system call will delete any prior extension (including the preceding period).

DQSCLOSE waits for completion of I/O operations taking place on the file (if any), empties output buffers, and frees any buffers associated with the CONNECTION.

```
CALL DQSCLOSE (connection, except$ptr);
```

INPUT PARAMETER

connection A WORD containing a token for a file CONNECTION that is currently open.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

The DQSCLOSE system call closes a connection that has been opened by the DQ\$OPEN system call. It performs the following steps:

1. It waits until all currently running I/O operations for the file are completed.
2. It ensures that any information in a partially filled output buffer is written to the file.
3. It releases any buffers associated with the file.
4. It closes the connection to the file. The connection is still valid, and can be re-opened if necessary.

Access Control

The Operating System performs no access checking before closing the connection.

DQ\$CREATE creates a new file and establishes a connection to that file.

```
connection = DQ$CREATE (path$ptr, except$ptr);
```

INPUT PARAMETER

path\$ptr A POINTER to a STRING that specifies the path of the file to be created.

OUTPUT PARAMETERS

connection A WORD in which the Operating System places a CONNECTION to the newly created file.

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This call creates a new file with the name you specify and returns the CONNECTION to your program. If a file of the same name already exists it is truncated (the data is destroyed).

To prevent accidentally destroying a file, issue DQ\$ATTACH before issuing DQ\$CREATE. If the file does not exist, you receive an exception code of E\$FNEXIST upon return from DQ\$ATTACH.

DQ\$DECODE\$EXCEPTION translates an exception code into an ASCII string.

```
CALL DQ$DECODE$EXCEPTION (except$code, buff$ptr, except$ptr);
```

INPUT PARAMETER

except\$code A WORD that contains the numeric exception code that is to be interpreted.

OUTPUT PARAMETERS

buff\$ptr A POINTER to a buffer (at least 81 bytes long) in which the system will return a STRING.

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

Your program provides the Operating System with the numeric value of an exception code, and the iRMX 86 Operating System returns the mnemonic and hex value of this code. For example, if you pass DQ\$DECODE\$EXCEPTION a value of 2 in **except\$code**, the system will return the following string:

0002: E\$MEM

The hex values and mnemonics for exception codes are listed in Appendix A.

DQ\$DELETE eliminates an existing file.

```
CALL DQ$DELETE (path$ptr, except$ptr);
```

INPUT PARAMETER

path\$ptr A POINTER to a STRING that specifies the pathname of the file to be deleted.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

A program can use this system call to delete a file. This system call will mark for deletion the specified file. This means that the system may actually postpone deletion if there are other connections to the file and delete the file only when all connections are closed and detached.

DQ\$DETACH breaks the connection established by DQ\$ATTACH or DQ\$CREATE.

```
CALL DQ$DETACH (connection, except$ptr);
```

INPUT PARAMETER

connection A WORD containing a token for the file CONNECTION to be deleted.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call deletes a file CONNECTION. If the CONNECTION is open, the DQ\$DETACH system call automatically closes it first (see DQ\$CLOSE). DQ\$DETACH will also delete the file if it has been marked for deletion and this is the last CONNECTION to the file.

DQSEXIT returns control from your program to the Operating System.

```
CALL DQSEXIT (end$code);
```

INPUT PARAMETERS

end\$code A WORD containing the encoded reason for termination of the program. You must include this code, but currently the iRMX 86 Operating System does not check this value. The standard codes are:

<u>VALUE</u>	<u>INTERPRETATION</u>
0	Termination was normal.
1	Warning messages were issued.
2	Errors were detected.
3	Fatal errors were detected.
4	The job was aborted.

DESCRIPTION

DQSEXIT terminates a program. All connections are detached, all files are closed, and any memory allocated to the program with DQ\$ALLOCATE is returned to the memory pool.

Calling DQSEXIT cannot result in an exception code.

DQ\$FREE returns to the Operating System a segment of memory acquired earlier by DQ\$ALLOCATE.

```
CALL DQ$FREE (base$addr, except$ptr);
```

INPUT PARAMETER

base\$addr A SELECTOR containing the base address of the segment that is to be deleted. This is the base address SELECTOR returned to your program by DQ\$ALLOCATE.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

The DQ\$FREE system call returns the specified segment to the memory pool from which it was allocated.

The DQ\$GET\$ARGUMENT system call is used to return successive arguments from a command line.

```
delimit$char = DQ$GET$ARGUMENT (argument$ptr, except$ptr);
```

INPUT PARAMETER

argument\$ptr A POINTER to a buffer in which the system will return the argument string. The length of the string (in bytes) is stored in the first byte of this area. The buffer must be at least 81 bytes long.

OUTPUT PARAMETERS

delimit\$char This is a single BYTE in which the system returns the delimiter character.

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

GET\$ARGUMENT is called to get successive arguments from a command line. The command line may be the same one that invoked the program containing this call. But if the UDI system call DQ\$SWITCH\$BUFFER is called before DQ\$GET\$ARGUMENT, the command line can be anywhere that you specify.

A delimiter is returned only if the exception code is zero. The following delimiters are recognized by the iRMX 86 Operating System:

,) (= # ! \$ % \ + - > < ~

as well as a space (), the grave accent (`), and any characters with hexadecimal values between 0 and 20H.

The Operating System will strip out ampersands (&) and semi-colons (;).

Before your program runs, the Operating System Command Line Interpreter (CLI) pre-edits the command line to remove comments and continuation characters. The Operating System also makes the following changes to the command line:

- Multiple adjacent blanks separating two arguments are treated as one blank. One or more blanks adjacent to any other delimiter are removed. A tab is treated as a blank and returned as a blank.
- Lower case characters are converted to upper case unless part of a quoted string.
- Strings enclosed within a matching pair of single or double quotes are considered literals. The enclosing quotes are not returned as part of the argument.

EXAMPLE

The following example illustrates the arguments and delimiters returned by successive calls to DQ\$GET\$ARGUMENT. The ARGUMENT LENGTH value is in the first byte of the string returned, the contents of each string is listed in the column ARGUMENT VALUE, and the delimiter returned in the byte delimit\$char is in the column DELIMITER.

Note that the last delimiter for each example is a carriage return (CR); this is how a program determines that there are no more arguments in the command line.

Table 4-2. Command Parsing Example

PLM86 LINKER.PLM PRINT(:LP:) NOLIST			
	ARGUMENT		
	<u>LENGTH</u>	<u>VALUE</u>	<u>DELIMITER</u>
	8	PLM86	(space)
	10	LINKER.PLM	(space)
	5	PRINT	(
	4	:LP:)
	6	NOLIST	CR

The DQ\$GET\$CONNECTION\$STATUS system call returns information about a file.

```
CALL DQ$GET$CONNECTION$STATUS (connection, info$ptr, except$ptr);
```

INPUT PARAMETER

connection A WORD containing a token for the CONNECTION whose status is desired.

OUTPUT PARAMETERS

info\$ptr A POINTER to a structure in which the Operating System will place the status information. The structure of info\$ptr should be:

```
DECLARE INFO STRUCTURE
      (OPEN            BYTE,
       ACCESS         BYTE,
       SEEK            BYTE,
       FILE$PTR$LOW    WORD,
       FILE$PTR$HIGH   WORD);
```

These fields are interpreted as follows:

OPEN 1 if connection is open, otherwise 2.

ACCESS Access privileges of the connection. The right is granted if the corresponding bit is set.

<u>BIT</u>	<u>ACCESS</u>
0	delete
1	read
2	write
3	update (read and write)

SYSTEMCALLS

info\$ptr (continued)

SEEK Types of seek supported.

VALUE	MEANING
0	no seek allowed
3	seek forward and backward

Values of 1 and 2 are not meaningful to the iRMX 86 Operating System.

FILE\$PTR\$HIGH These two items together form a 4-byte unsigned
FILE\$PTR\$LOW integer that indicates the current position in
the file. The position is expressed as the
number of bytes from the beginning of the
file, the first byte being byte 0 (zero).
This field is undefined if the file is not
open or if backward seek is not supported by
the device (for example, the printer cannot be
rewound).

except\$ptr A POINTER to a WORD where the system places the
condition code. Condition codes are described in
Appendix A.

DESCRIPTION

DQ\$GET\$CONNECTION\$STATUS is used to obtain information about a file CONNECTION. For example, you can use the system call if your program has performed a number of read or write operations and it is necessary to determine where the file pointer is now located.

DQ\$GET\$EXCEPTION\$HANDLER returns the address of the current exception handler.

```
CALL DQ$GET$EXCEPTION (address$ptr, except$ptr);
```

OUTPUT PARAMETERS

address\$ptr A POINTER to a four-byte area that the Operating System fills with a long pointer to the entry point of the current exception handler. A long pointer has the form:

```
DECLARE LONG$P STRUCTURE  
(LONG$OFFSET WORD,  
LONG$BASE WORD);
```

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

DQ\$GET\$EXCEPTION\$HANDLER is an informative system call that returns to your program the address of the current exception handler. This is the address specified in the last call to DQ\$TRAP\$EXCEPTION, if it has been called, otherwise the value returned is the address of the system default exception handler.

This routine always returns a four-byte pointer, even if called from a program compiled under the SMALL model of segmentation.

DQ\$GET\$EXCEPTION\$HANDLER is used in conjunction with DQ\$TRAP\$EXCEPTION and DQ\$DECODE\$EXCEPTION. See the descriptions of these calls for more information.

DQ\$GET\$SIZE returns the size of an allocated memory segment.

```
size = DQ$GET$SIZE (base$addr, except$ptr);
```

INPUT PARAMETER

base\$addr A SELECTOR containing the base address of a block of memory that was allocated with the DQ\$ALLOCATE call. This is the same address that is returned by DQ\$ALLOCATE when the segment was allocated.

OUTPUT PARAMETERS

size A WORD which the Operating System sets as follows:

- if not zero, contains the size, in bytes, of the segment identified by the base\$addr parameter
- if zero, indicates that the size of the segment is 65536 (64K) bytes.

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

The GET\$SIZE system call returns the size, in bytes, of a segment. You identify the segment of memory with a base address SELECTOR that is returned by the DQ\$ALLOCATE system call when the segment is allocated.

The size of the segment may not be exactly what you requested with the DQ\$ALLOCATE call. The Operating System allocates memory in 16-byte paragraphs. If you request a segment whose size is not a multiple of 16, the system increases the size to the next 16-byte boundary. This larger size is reflected in the size returned by DQ\$GET\$SIZE.

DQ\$GET\$SYSTEM\$ID returns a string that identifies the operating system.

```
CALL DQ$GET$SYSTEM$ID (id$ptr, except$ptr);
```

OUTPUT PARAMETERS

id\$ptr	POINTER to a 21-byte buffer in which the Operating System will place a STRING identifying the Operating System.
except\$ptr	A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call returns the following STRING:

iRMX 86

DQ\$GET\$TIME returns the current date and time in character format.

```
CALL DQ$GET$TIME (buff$ptr, except$ptr);
```

OUTPUT PARAMETERS

buff\$ptr A POINTER to a buffer in which the Operating System returns the current date and time. The structure of the buffer should be:

```
DECLARE DT STRUCTURE
          (DATE (8) BYTE,
          TIME (8) BYTE);
```

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call returns the current date and time, each as a series of bytes. DATE has the form MM/DD/YY for month, day, and year. The two slashes (/) are in the third and sixth bytes. For example, the date January 15th of 1982 would be returned as:

```
01/15/82
```

TIME has the form HH:MM:SS for hours, minutes, and seconds, with separating colons (:). The value for hours ranges from 0 through 23. For example, the time 20 seconds past 3:12 PM would be returned as:

```
15:12:20
```

The DQ\$OPEN system call is used to inform the Operating System how your program is going to access a file, and to identify the buffers you will use.

CALL DQ\$OPEN (connection, access, num\$buf, except\$ptr);

INPUT PARAMETERS

connection A WORD containing a token for the file CONNECTION to be opened.

access A BYTE telling how your program is going to use the CONNECTION. You should set the BYTE as follows:

<u>Value</u>	<u>Meaning</u>
1	Read only
2	Write only
3	Update (both reading and writing)

num\$buf A BYTE containing the number of buffers that you want the Operating System to allocate for this connection.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call prepares a connection for read, write, and seek commands. Your program can have up to twenty connections open at one time, if there is sufficient memory.

DESCRIPTION (continued)

DQ\$OPEN:

1. Creates the number of buffers requested.
2. Sets the connection's file pointer to zero. This is the pointer that tells the Operating System where in the file to perform an operation.
3. Starts reading ahead if num\$buf is greater than zero and the access parameter is "Read only" or "Update."

Selecting Access Rights

The system will not allow your program to read using a connection open for writing only, nor to write using a connection open for reading only. If you are not certain how the connection will be used, specify both reading and writing.

Selecting the Number of Buffers

The process of deciding how many buffers to allocate is based on three considerations -- compatibility, memory, and performance.

COMPATIBILITY. If you expect to run your program on other systems using UDI, you should use no more than two buffers.

MEMORY. The amount of memory used for buffers is directly proportional to the number of buffers. So you can save memory by using fewer buffers.

PERFORMANCE. The performance consideration is more complex. Up to a certain point, the more buffers you allocate, the faster your program can run. The actual break-even point, the point where more buffers don't improve performance, depends on many variables. Be aware that in order to overlap I/O with computation, you must specify at least two buffers. If performance is not at all important and memory is, use zero buffers.

Specifying zero buffers means that no buffering should occur; each DQ\$READ or DQ\$WRITE should result in a physical I/O operation. Interactive programs should open :CI: and :CO: with num\$buf set to zero to eliminate buffering.

If you normally seek before doing a read or write, num\$buf should be 1.

The DQ\$OVERLAY system call is invoked by a root module to load an overlay module.

```
CALL DQ$OVERLAY (name$ptr, except$ptr);
```

INPUT PARAMETER

name\$ptr A POINTER to a STRING that contains the name of an overlay module. The name must be in upper-case.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call is invoked by a root module whenever the root module wishes to load an overlay module.

If your assembly language or PL/M-86 programs use the DQ\$OVERLAY procedure, you should take care to ensure that you link the UDI library to your program correctly. The iAPX 86, 88 FAMILY UTILITIES USER'S GUIDE contains an example of linking an overlay program. This example lists a two-step link process, as follows:

1. Link the root and each of the overlays separately, specifying the OVERLAY control, but not the BIND control, in each LINK86 command.
2. Link all the output modules together in one module, specifying the BIND control, but not the OVERLAY control.

This is the same process that you should use when linking your iRMX 86 overlay programs. However, you must ensure that you link the entire UDI library to the root portion of the program and not to any of the overlays. To do this, use the INCLUDE control to include the UDI externals file (SYSTEM/UDI/UDI.EXT) with the assembly or compilation of the root portion of the program. By including this file with the root, you make external references to all UDI routines from that root. Then

DESCRIPTION (continued)

when you link the root to the UDI library, LINK86 pulls in all of the UDI routines, not just the ones called in the root. Since you are linking the UDI library to the root only, this prevents you from having unsatisfied external references when you link the root to the overlays.

For example, suppose your program consists of three files, ROOT.OBJ, OV1A.OBJ, and OV2A.OBJ, the root and overlay files, respectively. You have compiled these program modules with the PL/M-86 compiler and included the UDI externals file UDI.EXT with the compilation of the root. Assuming that LINK86 resides on the default logical device in the default directory, and that the object files reside in the directory PROG, the following LINK86 commands will link the overlay program and produce an executable module. This happens in two steps.

1. The first three LINK86 commands separately link the root and overlay portions of the program. The root portion of the program is linked to the UDI library (underlined entries are your commands).

```
-LINK86 PROG/ROOT.OBJ,      &  
**SYSTEM/UDI/URXLRG.LIB OVERLAY
```

```
IRMX 86 8086 LINKER Vx.y
```

```
-LINK86 PROG/OV1A.OBJ OVERLAY(OVERLAY1)
```

```
IRMX 86 8086 LINKER Vx.y
```

```
-LINK86 PROG/OV2A.OBJ OVERLAY(OVERLAY2)
```

2. The next LINK86 command links together in one module all the output modules produced in the first step.

```
-LINK86 PROG/ROOT.LNK,      &  
**PROG/OV1A.LNK,           &  
**PROG/OV2A.LNK           &  
**TO PROGRAM1 BIND MEMPOOL(+2000H)
```

The DQ\$READ moves a number of bytes from a file to a buffer. Your calling program must specify the connection, the number of bytes, and the buffer to receive the information.

```
bytes$read = DQ$READ (connection, buff$ptr, bytes$max,  
                      except$ptr);
```

INPUT PARAMETERS

connection	A WORD containing a token for the connection to the file. This connection must be open for reading or for both reading and writing, and the file pointer of the connection must point to the first byte to be read.
buff\$ptr	A POINTER to a buffer that will receive the data that the Operating System reads from the file.
bytes\$max	A WORD containing the maximum number of bytes you expect to read from the file.

OUTPUT PARAMETERS

bytes\$read	A WORD containing the actual number of bytes read. This number is always equal to or less than the bytes\$max.
except\$ptr	A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call reads a collection of contiguous bytes from the file associated with the connection. These bytes are placed in a buffer specified by the calling program.

DESCRIPTION (continued)

The Buffer

The `buff$ptr` parameter tells the Operating System where to place the bytes after they are read. This is a buffer you create, and if it is not long enough, the Operating System overwrites the area beyond the buffer.

Number of Bytes Read

The number of bytes that your program requests is the maximum number of bytes that the Operating System places in the buffer. However, there are two circumstances under which the system reads fewer bytes.

- First, if the Operating System detects an end of file before reading the number of bytes requested, it will return only those bytes preceding the end of file. The `bytes$read` parameter can be less than the `bytes$desired` parameter, and no exceptional condition will be indicated.
- Second, if an exceptional condition does occur during the reading operation, information in the buffer and the value of the `bytes$read` parameter are meaningless.

Access Control

If the connection is not opened for reading or both reading and writing, the Operating System returns an exceptional condition.

The DQ\$RENAME system call changes the pathname of a file.

```
CALL DQ$RENAME (path$ptr, new$path$ptr, except$ptr);
```

INPUT PARAMETERS

- path\$ptr A POINTER to a STRING that specifies the pathname for the file to be renamed.
- new\$path\$ptr A POINTER to a STRING that specifies the new path for the file. This path cannot refer to an existing file.

OUTPUT PARAMETER

- except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call allows your programs to change the pathname for a file or a directory. Be aware that when you rename a directory, you are changing the pathnames of all files contained in the directory. When you rename a file to which a connection exists (this is valid) the connection to the renamed file remains established.

Your program can change any aspect of the pathname so long as the file or directory remains on the same volume.

DQ\$SEEK changes the file position pointer.

```
CALL DQ$SEEK (connection, mode, hi$move$count, lo$move$count,
              except$ptr);
```

INPUT PARAMETERS

connection A WORD containing a token for an open connection whose file pointer you wish to move.

hi\$move\$count These two WORDS combine to form a 32-bit integer
lo\$move\$count that tells the Operating System how many bytes to move the file pointer.

mode A BYTE containing a value that controls the nature of the movement of the file pointer. Any of the following values are valid:

<u>Mode</u>	<u>Meaning</u>
-------------	----------------

- | | |
|---|--|
| 1 | Move the pointer backward by the specified move count. If the move count is large enough to position the pointer past the beginning of the file, set the pointer to the first byte (position zero). |
| 2 | Set the pointer to the position specified by the move count. Position zero is the first position in the file. Moving the pointer beyond the end of the file is valid. |
| 3 | Move the file pointer forward by the specified move count. Moving the pointer beyond the end of file is valid. |
| 4 | First move the pointer to the end of the file and then move it backward by the specified move count. If the specified move count would position the pointer beyond the front of the file, set the pointer to the first byte in the file (position zero). |

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

When performing random I/O, your programs must use this system call to position the file pointer before using the DQ\$READ, DQ\$TRUNCATE, or DQ\$WRITE system calls. The location of the file pointer tells the Operating System where in the file to begin reading, truncating, or writing information. If your program is performing sequential I/O on a file, they do not need to use this system call.

As mentioned previously, it is legitimate to position the file pointer beyond the end of file. If your program does this and then invokes the DQ\$READ system call, the Operating System behaves as though the read operation began at the end of file.

Also, it is possible to invoke the DQ\$WRITE system call with the file pointer beyond the end of the file. If your program does this, the Operating System attempts to expand the file. Be aware that if you expand your file in this manner, the expanded portion of the file will contain undefined information.

DQSSPECIAL specifies whether line editing is to be performed by the Operating System on console input.

```
CALL DQSSPECIAL (mode, conn$ptr, except$ptr);
```

INPUT PARAMETERS

mode A BYTE used to change the mode of terminal input. The values are:

- Transparent 1
- Line editing (default)
- Transparent 3

Each of these types is explained in the description.

conn\$ptr A POINTER to a token for the CONNECTION to the file. The CONNECTION must be a connection to :CI: established by DQ\$ATTACH.

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call is used to change the technique by which your program receives input from a console input device. The default mode is line editing, but by using DQSSPECIAL you can change from line editing to one of the transparent modes, or back to line editing.

The meanings of the type parameter are as follows.

DESCRIPTION (continued)

<u>Value</u>	<u>Meaning</u>
1	<u>Transparent 1.</u> Interactive programs often need to obtain characters from the console exactly as they are typed. This is made possible by transparent mode. In transparent mode, all characters are placed in the buffer specified by the call to DQ\$READ. (The only exception are CTRL/C, which will terminate the program, and CTRL/D, which has no effect on the system.) The Operating System returns control to the calling program when the number of characters typed equals the number of characters specified in the DQ\$READ system call.
1 2	<u>Line editing.</u> This means that the console operator has the opportunity to correct typing errors. Data from the console is not actually returned by a call to DQ\$READ until the operator types a carriage return. The Operating System removes editing characters (such as the backspace) from the input, and also adds a line feed to the final carriage return if buffer space permits.
3	<u>Transparent 3.</u> This is nearly the same as Transparent 1 mode, except that in Tranparent 3 mode the Operating System returns control to your program immediately after the DQ\$READ call, whether or not any characters have actually been typed since the last DQ\$READ. If no characters have been typed, this will be indicated by the bytes\$read parameter of the DQ\$READ call. Characters that are typed between successive calls to read the terminal are held in a special Operating System buffer called the "type-ahead" buffer.

SYSTEMCALLS

DQ\$SWITCH\$BUFFER is used with DQ\$GET\$ARGUMENT to get arguments from a command line contained within your program.

```
offset = DQ$SWITCH$BUFFER (buff$ptr, except$ptr);
```

INPUT PARAMETERS

buff\$ptr A POINTER to a STRING containing the text to be parsed.

OUTPUT PARAMETERS

offset A WORD that the Operating System sets equal to the number of bytes from the beginning of the buffer to first character in the next argument in the buffer.

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

DQ\$SWITCH\$BUFFER is used to point to a command line other than the line that invoked this program. Typically, you will first call DQ\$SWITCH\$BUFFER, and then make a series of calls to DQ\$GET\$ARGUMENT. Each call to DQ\$GET\$ARGUMENT fetches an argument from the line pointed to by buff\$ptr.

DQ\$SWITCH\$BUFFER can also be used as an informative system call if you need to know the location of the next argument in the line. If you call DQ\$SWITCH\$BUFFER without changing buff\$ptr, the value returned in offset is the number of bytes from the beginning of the command line to the first character in the next argument.

The parameter offset will be zero (0) upon return from the first call to DQ\$SWITCH\$BUFFER.

DQ\$TRAP\$EXCEPTION substitutes an alternate exception handler for the default exception handler provided by the operating system.

```
CALL DQ$TRAP$EXCEPTION (address$ptr, except$ptr);
```

INPUT PARAMETERS

address\$ptr the address of a four-byte area containing a long pointer to the entry point of the alternate exception handler. A long pointer has the form:

```
DECLARE LONG$P STRUCTURE
      (LONG$OFFSET WORD,
       LONG$BASE   WORD);
```

OUTPUT PARAMETER

except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

DQ\$TRAP\$EXCEPTION is used to inform the Operating System that when an exceptional condition occurs, the Operating System is to pass control to your exception handler. An exceptional condition is defined as a return from a system call with a condition code other than E\$OK (see Appendix A for exception code meanings).

See the section EXCEPTION-HANDLING SYSTEM CALLS at the beginning of this chapter for an explanation of the conditions of the stack when your exception handler receives control.

DQ\$TRUNCATE removes information from the position of the file pointer to the end of the file.

```
CALL DQ$TRUNCATE (connection, except$ptr);
```

INPUT PARAMETER

connection	A WORD containing a token for a CONNECTION to the named data file that is to be truncated. The file pointer of this CONNECTION tells the Operating System where to truncate the file. The BYTE indicated by the pointer is the first byte to be dropped from the file.
------------	--

OUTPUT PARAMETER

except\$ptr	A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.
-------------	--

DESCRIPTION

This system call truncates a file at the current setting of the file pointer and frees all space beyond the pointer. If the pointer is at or beyond the end of file, no truncation will be performed. Unless the file pointer is already where you want it, your program should use the DQ\$SEEK system call to position the pointer before using the DQ\$TRUNCATE system call.

The CONNECTION should have write, or read and write access rights, established when the connection is opened.

The DQ\$WRITE system call moves a collection of bytes from a buffer into a file.

```
CALL DQ$WRITE (connection, buff$ptr, count, except$ptr;
```

INPUT PARAMETERS

- connection A WORD containing a token for the CONNECTION to the file in which the information is to be written.
- buff\$ptr A POINTER to a collection of contiguous bytes that are to be written to the specified file.
- count A WORD containing the number of bytes to be written from the buffer to the file.

OUTPUT PARAMETERS

- except\$ptr A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

This system call causes the Operating System to write the specified number of bytes from the buffer to the file.

Access Control

In order to write information into a file. The file must be open for writing, or for reading and writing (update access). Whenever your program attempts to write over information in a file via a connection that does not have update access, the Operating System does not write any data to the file but returns an exception code. The description of DQ\$OPEN explains how access is established.

DESCRIPTION (continued)

Number of Bytes Written

Occasionally, the Operating System writes fewer bytes than requested by the calling program. This happens under two circumstances. The first circumstance is when the Operating System encounters an I/O error (E\$IO exception code is returned).

The second circumstance is when the volume to which your program is writing becomes full. The Operating System informs your program of this condition by returning an E\$SPACE exception code.

Where the Bytes Are Written

The Operating System writes the bytes starting at the location specified by the connection's file pointer. (The pointer indicates where the first byte is to be written.) The pointer is updated as the bytes are written. After the writing operation is completed, the file pointer points to the byte immediately following the last byte written.

If your program must reposition the file pointer before writing, it can do so by using the DQ\$SEEK system call.

SYSTEMS

EXAMPLE PROGRAM

This program provides an example of UDI system calls.

```

$compact
$optimize(3)
/*.....
 * Program UPPER
 *
 * This program demonstrates the use of UDI file handling:
 * and command line parsing system calls. The program reads an input
 * file of characters, and converts all lower-case alphabetic
 * characters to upper-case. The converted date is written to a
 * second file.
 *
 * UPPER expects the command line that invokes it to be of the form:
 *
 * UPPER infile [TO outfile]
 *
 * (If "TO outfile" is not specified, :CO: is assumed.)
 *.....
 */

```

upper: DO;

\$include(system/udi/udi.ext) /* This file is described at the beginning
of the chapter */

DECLARE

```

CR      LITERALLY 'ODH',
LF      LITERALLY 'OAH',
E$OK    LITERALLY 'O';

```

DECLARE

```

co$comm      WORD;

```

SYSTEM CALLS

EXAMPLE PROGRAM (continued)

\$subtitle('check\$exception')

```
/*.....  
* Procedure to check an exception code. If the exception code is  
* not E$OK, print a message and exit.  
*.....  
*/
```

check\$exception: PROCEDURE(exception, info\$p) REENTRANT;

 DECLARE

```
    exception WORD,  
    info$p      POINTER,  
    info      BASED info$p STRUCTURE(  
      count      BYTE,  
      char(1)    BYTE),  
    exc$buf     STRUCTURE(  
      count      BYTE,  
      char(80)  BYTE),  
    dummy      WORD;
```

 IF exception <> E\$OK THEN

 DO;

 CALL dq\$decode\$exception(exception, @exc\$buf, @dummy);

 CALL dq\$write(co\$conn, @exc\$buf.char, exc\$buf.count,
 @dummy);

 CALL dq\$write(co\$conn, @(': '), 2, @dummy);

 CALL dq\$write(co\$conn, @info.char, info.count, @dummy);

 CALL dq\$write(co\$conn, @(CR, LF), 2, @dummy);

 CALL dq\$exit(3);

 END;

END check\$exception;

EXAMPLE PROGRAM (continued)

```

$subtitle('Main')
/*.....
*
*                               --- MAIN PROGRAM ---
*
*.....
*/

    DECLARE st WORD;

    DECLARE
        in$name(50)          BYTE,
        out$name(50)        BYTE,
        in$conn              WORD,
        out$conn             WORD,
        delim                BYTE;

    DECLARE
        buffer(1024)        BYTE,
        in$bp                POINTER,
        in$char              BASED in$bp BYTE,
        nextchar             BASED in$bp (2) BYTE,
        in$count             WORD,
        n$write              WORD,
        i                    WORD;

/*.....
*   Create a connection to :CO: (console output)
*.....
*/

co$conn = dq$create(@4, ':CO:'), @st);

CALL dq$open(co$conn, 2, 0, @st);

/*.....
*   Ignore the name of the program
*.....
*/

delim = dq$get$argument(@buffer, @st);
CALL check$exception(st, 0);
IF delim = CR THEN
    CALL dq$exit(0);

```

UDI SYSTEM CALLS

EXAMPLE PROGRAM (continued)

```

/*.....
 *   Attach the input file, and open it.
 *.....
 */

delim = dq$get$argument(@in$name, @st);
CALL check$exception(st, 0);

in$conn = dq$attach(@in$name, @st);
CALL check$exception(st, @in$name);

CALL dq$open(in$conn, 1, 2, @st);
CALL check$exception(st, @in$name);

/*.....
 *   Make ready the output file
 *.....
 */

IF delim <> CR THEN
    DO;
        delim = dq$get$argument(@buffer, @st);
        CALL check$exception(st, 0);
        IF (delim = CR) OR
            (buffer(0) <> 2) OR
            (buffer(1) <> 'T') OR
            (buffer(2) <> 'O') THEN
            DO;
                CALL dq$write(co$conn, @('Invalid output file',
                    CR, LF), 21, @st);
                CALL dq$exit(3);
            END;

        delim = dq$get$argument(@out$name, @st);
        CALL check$exception(st, 0);

        out$conn = dq$create(@out$name, @st);
        CALL check$exception(st, @out$name);

        CALL dq$open(out$conn, 2, 2, @st);
        CALL check$exception(st, @out$name);
    END;
ELSE
    out$conn = co$conn;

```

EXAMPLE PROGRAM (continued)

```

/*.....
 *   Read from input, convert, and write to output
 *.....
 */

DO WHILE 1;
  in$count = dq$read(in$conn, @buffer, size(buffer), @st);
  CALL check$exception(st, @in$name);
  IF in$count = 0 THEN
    GOTO end$of$file;

  DO i=0 TO in$count-1;
    IF (buffer(i) >= 'a') AND (buffer(i) <= 'z') THEN
      buffer(i) = buffer(i) + 'A'-'a';
    END;

    CALL dq$write(out$conn, @buffer, in$count, @st);
    CALL check$exception(st, @out$name);
  END;
end$of$file:

/*.....
 *   Close input and output files and exit
 *.....
 */

CALL dq$close(in$conn, @st);
CALL check$exception(st, @in$name);

CALL dq$close(out$conn, @st);
CALL check$exception(st, @out$name);

CALL dq$exit(0);

END upper;

```

SYSTEMCALLS

CHAPTER 5. PREPARING YOUR HARDWARE

This chapter describes how to prepare the hardware devices on which the iRMX 86 PC Operating System runs, (see Figure 5-1). The iRMX 86 PC product is a version of the iRMX 86 Operating System that has been prepared by Intel to run in the hardware environment described here.

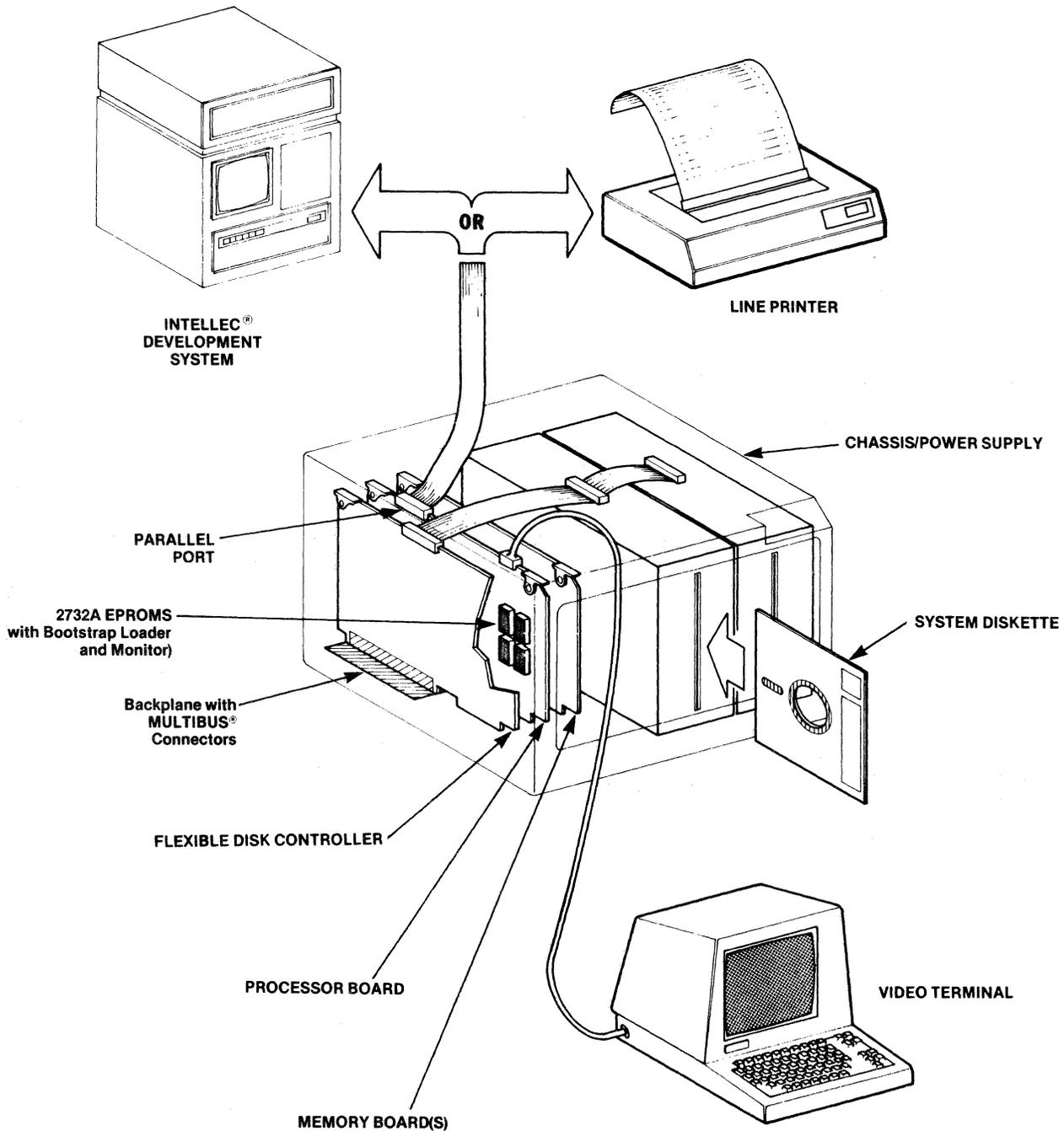


Figure 5-1. The iRMX™ 86 PC Hardware

PREPARING YOUR HARDWARE

This chapter is organized as follows:

- THE iRMX 86 PC HARDWARE ENVIRONMENT. A section describing hardware on which the iRMX 86 PC will run.
- MODIFYING BOARDS. A section describing how to modify your iAPX 86-based Single Board Computer, and how to modify the iSBC 208 Disk Controller board.
- CONVENIENCE CHARTS. To make the process of preparing your hardware easier, we have included (for each board described in this chapter) a single-page condensed summary of modifications required for that board. This is so that you can remove the page and refer to it as you actually install required jumpers and devices. These charts are the last four pages (two physical pages) in this chapter.

THE iRMX 86 PC HARDWARE ENVIRONMENT

The iRMX 86 PC Operating System runs on the following hardware components (shown in Figure 5-1):

- An Intel iSBC 86 processor board.
- An iSBC 208 Flexible Disk Drive Controller with at least two drives (you can connect as many as four drives).
- A video terminal connected to the computer board serial port. The software assumes that this terminal is RS232C-compatible, and set to 9600 baud full-duplex, with no parity checking (the computer ignores parity from the terminal, and sets the parity bit to zero (0) on data to the terminal).
- An appropriate chassis/power-supply unit with a Multibus-compatible backplane.

In addition, you can also have either a line printer or an iSBC 957B package (they both use the same parallel port on the computer board). The iSBC 957B package allows you to connect your system directly to an Intellec Microcomputer Development System. Neither the line printer nor the iSBC 957B package is required to run the Operating System.

SINGLE BOARD COMPUTER

The iRMX 86 PC Operating System runs on any of these Single Board Computers: iSBC 86/12A, iSBC 86/14, or iSBC 86/30.

PREPARING YOUR HARDWARE

FLEXIBLE DISKETTE CONTROLLER AND DRIVES

The iSBC 208 Controller will handle up to four drives. Two drives are required to run the iRMX 86 PC System. The iSBC 208 Controller will accept drives and diskettes of many recording formats: single- and double-density, single- and double-sided, and sector sizes of 128, 256, and 1024 bytes. The diskettes are soft-sectored, which means that you must format new diskettes using the FORMAT command described in Chapter 3. The iRMX 86 PC System is delivered on double-density, single-sided diskettes having 256 bytes-per-sector.

Physical names that must be used with the iSBC 208 Controller for various disk characteristics and drives are listed in Table 5-1. Although only drive number 0 is listed for each type, the controller and the iRMX 86 PC Operating System support up to four drives (for example, AFD0, AFD1, AFD2, and AFD3). You specify these names when using the iRMX 86 PC Command ATTACHDEVICE described in Chapter 3.

Table 5-1. iSBC® 208 Physical Names

Device Names	Device Type	Sides	Density	Bytes per Sector
AFO	208 Shugart SA800	1	Single	128
AFDO	208 Shugart SA800	1	Double	256
AFDDO	208 Shugart SA850	2	Double	256
AFDXO	208 Shugart SA850	2	Double	1024

MEMORY

The iRMX 86 PC Operating System requires at least 192K bytes of memory to run. In addition, you will need memory sufficient to run your programs, system utilities, and language processors. Some memory is on-board the iSBC 86 Single Board Computer, with the remainder on one or more memory boards. On-board memory may include a RAM expansion module; with a RAM expansion module the iSBC 86/30 has enough memory to run the Operating System.

LINE PRINTER

You can use any line printer that recognizes the Centronics signal/pin standard. The line printer is connected to the parallel port on the processor board you use.

PREPARING YOUR HARDWARE

Table 5-2 shows the signals that are present on pins at the:

- 50-Pin iSBC Connector: The Single Board Computer parallel port connector (J1).
- 50-Pin Edge Connector: A standard 50-pin edge connector used as a cable-end; mates to the iSBC Connector described in 1.
- 30-Pin Connector: The Centronics-standard plug and connector at the line printer.

Table 5-2. Line Printer Pin Assignments

50-Pin iSBC Connector	50-Pin Edge Connector	-- CENTRONICS-Standard --	
		30-Pin Connector	Signal
24	23	1	Character strobe to printer
26	25	13	SLCT (Select)
28	27	12	Paper Out
30	29	10	ACKNOWLEDGE from printer
34	33	9	Data Bit 7
36	35	8	Data Bit 6
38	37	7	Data Bit 5
40	39	6	Data Bit 4
42	41	5	Data Bit 3
44	43	4	Data Bit 2
46	45	3	Data Bit 1
48	47	2	Data Bit 0
<p>NOTES:</p> <p>iSBC Connector: All odd pin numbers are grounded.</p> <p>Edge Connector: All even-numbered pins grounded.</p> <p>Centronics LP Connector: Pins 19-29 Protective grounds Pin 16 Logic Ground Pin 17 Chassis Ground</p>			

PREPARING YOUR HARDWARE

iSBC 957B PACKAGE

You can use the parallel port for copying files to and from an Intellect Development System using an iSBC 957B hardware/software package. This chapter describes changes to your Single Board Computer required to support the iSBC 957B package. Refer to the USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE for information about how to connect your system to a Development System.

MODIFYING BOARDS

This section describes how to modify your Single Board Computer, and the iSBC 208 Disk Controller Board. There is a section devoted to each of the following:

- iSBC 208 Flexible Diskette Controller
- iSBC 86/12A Single Board Computer
- iSBC 86/14 Single Board Computer
- iSBC 86/30 Single Board Computer

In discussions of how to modify boards, the term "modify" means installing non-standard jumpers, and installing components on the board (such as the EPROMs that come as part of the iRMX 86 PC System). The discussions in this chapter assume: that you have a hardware reference manual for the board you are modifying, and that boards you modify have only the factory-installed jumpering in place.

To modify a Single Board Computer to support the iRMX 86 PC Operating System, you must know:

- Whether you are using the parallel port for a line printer or for an iSBC 957B package.
- Whether your board has an iSBC 337 Multimodule Numeric Data Processor ("NDP").
- Whether you are using a RAM expansion module on your Single Board Computer.
- Whether you are using serial bus priority resolution (the factory default in all cases) or parallel priority resolution.

The instructions that follow note when a particular jumper or device is affected by these variables.

Comments describing the effect of jumpers are very brief; more complete descriptions are in the appropriate hardware reference manual.

PREPARING YOUR HARDWARE

MODIFYING THE iSBC 208 CONTROLLER

Regardless which Single Board Computer you are using for your system, the iSBC 208 Disk Controller is jumpered the same way. The jumpers are shown in Table 5-3.

Table 5-3. iSBC[®] 208 Jumpers

Remove Jumper	Add Jumper	Function/Description
E45-E49 E77-E78	E79-E84 E41-E45	Interrupt level 5 16-bit I/O address decoding. Only if parallel bus priority resolution is used. (Factory sends board with this jumper removed, for serial bus priority resolution).
NOTES: The iSBC 208 16-bit I/O base address is 0000H (default). The controller comes from the factory supporting 8-inch flexible diskettes. The Operating System requires 8-inch diskettes.		

The controller is documented in the iSBX 208 FLEXIBLE DISK DRIVE CONTROLLER HARDWARE REFERENCE MANUAL.

PREPARING YOUR HARDWARE

MODIFYING THE iSBC 86/12A SINGLE BOARD COMPUTER

The following tables list the modifications necessary to support the iRMX 86 PC Operating System with an iSBC 86/12A Microcomputer.

Interrupt Level Jumpers

Table 5-4 summarizes jumpers that establish interrupt levels.

Table 5-4. Interrupt Jumpers for iSBC® 86/12A

Add Jumper	Function/Description
E81-E1	Interrupt Level 0, iSBC 337 (1)
E72-E89	Level 1, Non-Maskable Interrupt
E80-E84	Level 1, Line Printer
E79-E83	Level 2, System Clock (2)
E68-E76	Level 5, iSBC 208 (2)
E75-E82	Level 6, Terminal Driver (Read)
E74-E90	Level 7, Terminal Driver (Write)

NOTES:

(1) This jumper is for an iSBC 86/12A with PWA number of 142977-XXX. If an older version of an iSBC 86/12A is used, refer to the iSBC 337 MULTIMODULE NUMERIC DATA PROCESSOR HARDWARE REFERENCE MANUAL.

(2) Factory-installed jumpers.

Additional Jumpers

Table 5-5 summarizes additional jumpering of the iSBC 86/12A.

Table 5-5. Other iSBC® 86/12A Jumpers

Remove Jumper	Add Jumper	Function/Description
E125-E126	E51-E52	Clear To Send signal capability
E97-E98	E127-E128	Set dual-port RAM address
E151-E152	E12-E21	Only if iSBC 337 is not used
	E97-E99	Required by Monitor EPROMs
	E5-E6	Unpopulated memory or port time-out
		Only for parallel priority resolution

PREPARING YOUR HARDWARE

Parallel Port

Table 5-6 summarizes jumper setting for the iSBC 86/12A Parallel Port, which can be used for either a line printer or an iSBC 957B package.

Table 5-6. iSBC® 86/12A Parallel Port Jumpers

iSBC 957B		Line Printer	
Remove Jumper	Add Jumper	Remove Jumper	Add Jumper
E13-E14 E19-E20 E21-E25 E26-E27 E30-E31 E32-E33	E13-E27 E14-E30 E18-E31 E20-E33 E25-E31	E13-E14 E32-E33	E22-E32

Switch Settings

Table 5-7 Shows the settings for each position (segment) of Switch 1.

Table 5-7. iSBC® 86/12A Switch 1

Position	Setting	Position	Setting
1	ON	5	OFF
2	(1)	6	OFF
3	OFF	7	ON
4	OFF	8	OFF

NOTE: (1) Switch 2 must be OFF if you are using an iSBC 300 RAM Expansion Module, otherwise ON.

PREPARING YOUR HARDWARE

Devices

Table 5-8 describes the devices that must be installed on your iSBC 86/12A Board.

Table 5-8. iSBC® 86/12A Devices

Device	Part Number	Socket
2732A EPROM	144447-001	A28
2732A EPROM	144448-001	A29
2732A EPROM	144449-001	A46
2732A EPROM	144450-001	A47
iSBC 902 Resistor packs	4500645-01	A10, A12, A13
7438 IC	100908-001	A11 (1)
Status Adapter	1002129	A11 (2)
<p>NOTES:</p> <p>(1) If parallel port is used for line printer.</p> <p>(2) If parallel port is used for iSBC 957B package.</p>		

The iSBC 86/12A is documented in the iSBC® 86/12A Single Board Computer Hardware Reference Manual (Order Number 9803074).

PREPARING YOUR HARDWARE

MODIFYING THE iSBC 86/14 SINGLE BOARD COMPUTER

The following tables list modifications necessary to support the iRMX 86 PC Operating System with an iSBC 86/14 Single Board Computer.

Jumpers

Table 5-9 summarizes jumpers that establish interrupt level jumpers, Table 5-10 summarizes parallel port jumpers, and Table 5-11 shows all other jumpers that must be installed on the iSBC 86/14 Board.

Table 5-9. Interrupt Jumpers for iSBC® 86/14

Remove Jumper	Add Jumper	Function/Description
E144-E145	E165-E166 E145-E149 E132-E164 E147-E158 E151-E152 E153-E155 E134-E154	Interrupt Level 0, iSBC 337 Level 1, Non-Maskable Interrupt Level 1, Line Printer Level 2, System Clock (1) Level 5, iSBC 208 (1) Level 6, Terminal Driver (Read) Level 7, Terminal Driver (Write)
NOTE: (1) Factory-installed jumpers.		

Table 5-10. iSBC® 86/14 Parallel Port Jumpers

If iSBC 957B is used		If Line Printer is used	
Remove Jumper	Add Jumper	Remove Jumper	Add Jumper
E44-E53 E45-E54 E46-E55 E48-E57 E50-E59 E51-E60 E52-E61	E44-E59 E45-E50 E45-E54 E46-E51 E48-E53 E50-E52	E44-E53 E51-E60	E60-E63

PREPARING YOUR HARDWARE

Table 5-11. Other iSBC® 86/14 Jumpers

Remove Jumper	Add Jumper	Function/Description
E219-E225	E76-E77	Clear To Send signal capability.
E26-E27	E61-E62	Set dual-port RAM address
E33-E34		Only if iSBC 337 is not used.
		Enable Non-Maskable Interrupt
		Non-bus vector interrupt
	E36-E37	Selects 5MHz clock (1)
E111-E112	E124-E125	Selects 2732-type EPROM
	E112-E113	2732 EPROM address range
	E119-E120	If RAM Expansion Module <u>is</u> used
	E230-E231	If RAM expansion module <u>not</u> used
	E232-E233	Dual-port Ram Addressing
E210-E211		For parallel priority resolution
NOTE: (1) 5 MHz required if iSBC 337 NDP is used.		

Devices

Table 5-12 describes the devices that must be installed on your iSBC 86/14 Board.

Table 5-12. iSBC® 86/14 On-Board Devices

Device	Part Number	Socket
2732A EPROM	144447-001	U57
2732A EPROM	144448-001	U58
2732A EPROM	144449-001	U39
2732A EPROM	144450-001	U40
902 Resistor Packs	4500645-01	U18, U20, U21
7438 IC	100908-001	U19 If Line Printer used
Status Adapter	1002129	U19 If iSBC 957B used

The iSBC 86/14 Single Board Computer is documented in the iSBC® 86/14 AND iSBC® 86/30 SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL.

PREPARING YOUR HARDWARE

MODIFYING THE iSBC 86/30 SINGLE BOARD COMPUTER

The following tables list modifications necessary to support the iRMX 86 PC Operating System with an iSBC 86/30 Single Board Computer.

Jumpers

Table 5-13 summarizes jumpers that establish interrupt level jumpers, Table 5-14 summarizes parallel port jumpers, and Table 5-15 summarizes all other jumpers that must be installed on your iSBC 86/30 Board.

Table 5-13. Interrupt Jumpers for iSBC® 86/30

Remove Jumper	Add Jumper	Function/Description
E144-E145	E165-E166 E145-E149 E132-E164 E147-E158 E151-E152 E153-E155 E134-E154	Interrupt Level 0, iSBC 337 Level 1, Non-Maskable Interrupt Level 1, Line Printer Level 2, System Clock (1) Level 5, iSBC 208 (1) Level 6, Terminal Driver (Read) Level 7, Terminal Driver (Write)
NOTE: (1) Factory-installed jumpers.		

Table 5-14. iSBC® 86/30 Parallel Port Jumpers

If iSBC 957B is used		If Line Printer is used	
Remove Jumper	Add Jumper	Remove Jumper	Add Jumper
E44-E53 E45-E54 E46-E55 E48-E57 E50-E59 E51-E60 E52-E61	E44-E59 E45-E50 E45-E54 E46-E51 E48-E53 E50-E52	E44-E53 E51-E60	E60-E63

PREPARING YOUR HARDWARE

Table 5-15. Other iSBC® 86/30 Jumpers

Remove Jumper	Add Jumper	Function/Description
E219-E225	E76-E77	Clear To Send signal capability.
E26-E27	E61-E62	Set dual-port RAM address
E33-E34		Only if iSBC 337 is not used.
		Enable Non-Maskable Interrupt
		Non-bus vector interrupt
	E36-E37	Selects 5MHz clock (1)
	E124-E125	Selects 2732-type EPROM
E111-E112	E112-E113	2732 EPROM address range
	E119-E120	If RAM Expansion Module <u>is</u> used
	E232-E233	If RAM expansion module <u>not</u> used
E210-E211		For parallel priority resolution
NOTE: (1) 5 MHz required if iSBC 337 NDP is used.		

Devices

Table 5-16 describes the devices that must be installed.

Table 5-16. iSBC® 86/30 On-Board Devices

Device	Part Number	Socket
2732A EPROM	144447-001	U57
2732A EPROM	144448-001	U58
2732A EPROM	144449-001	U39
2732A EPROM	144450-001	U40
902 Resistor Packs	4500645-01	U18, U20, U21
7438 IC	100908-001	U19 If Line Printer used
Status Adapter	1002129	U19 If iSBC 957B used

The iSBC 86/30 Single Board Computer is documented in the iSBC® 86/14 AND iSBC® 86/30 SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL.

PREPARING YOUR HARDWARE

CONVENIENCE CHARTS

The next four pages are printed for your convenience in working with the the boards that you must modify. You are invited to remove these pages and use them as work guides. There is one page each for the following boards::

- iSBC 86/12A Single Board Computer
- iSBC 86/14 Single Board Computer
- iSBC 86/30 Single Board Computer
- iSBC 208 Flexible Disk Controller

The iSBC 208 page also includes the Centronics-to-parallel port information, in case you must connect a line printer to one of the computers.

PREPARING YOUR HARDWARE

Table 5-17. iSBC® 86/12A Jumpers (Condensed)

Remove Jumper	Add Jumper	Function/Description
	E81-E1 E72-E89 E80-E84 E79-E83 E68-E76 E75-E82 E74-E90	Interrupt Level 0, iSBC 337 (1) Level 1, Non-Maskable Interrupt Level 1, Line Printer Level 2, System Clock (factory-installed jumper) Level 5, iSBC 208 (factory-installed jumper) Level 6, Terminal Driver (Read) Level 7, Terminal Driver (Write)
E125-E126 E97-E98 E151-E152	E51-E52 E127-E128 E97-E99 E12-E21 E5-E6	Clear-To-Send signal capability. Set dual-port RAM address Required for Monitor in EPROMs Only if iSBC 337 is not used. Unpopulated memory or port time-out Only for parallel priority resolution
E13-E14 E19-E20 E21-E25 E26-E27 E30-E31 E32-E33	E13-E27 E14-E30 E18-E31 E20-E33 E25-E31	If parallel port is used for 957B
E13-E14 E32-E33	E22-E32	If parallel port is used for line printer
(1) Applies only to iSBC 86/12A with PWA of 142977-XXX		

Table 5-18. iSBC® 86/12A Devices (Condensed)

Device	Part Number	Socket
2732A EPROM	144447-001	A28
2732A EPROM	144448-001	A29
2732A EPROM	144449-001	A46
2732A EPROM	144450-001	A47
902 Resistor Packs	4500645-001	A10, A12, A13
7438 IC	100908	A11 Parallel port used for line printer
Status Adapter	1002129	A11 Parallel port used for iSBC 957B

Table 5-19. iSBC® 86/12A Switch 1 (Condensed)

Segment	Setting	Segment	Setting	
1	ON	5	OFF	* If iSBC 300 RAM expansion module is <u>not</u> used, this position ON
2	OFF (*)	6	OFF	
3	OFF	7	ON	
4	OFF	8	OFF	

PREPARING YOUR HARDWARE

Table 5-20. iSBC® 86/14 Jumpers (Condensed)

Remove Jumper	Add Jumper	Function/Description
E144-E145 E219-E225 E26-E27 E33-E34 E111-E112 E210-E211	E165-E166 E145-E149 E132-E164 E147-E158 E151-E152 E153-E155 E134-E154 E76-E77 E61-E62 E36-E37 E124-E125 E112-E113	Interrupt Level 0, iSBC 337 Level 1, Non-Maskable Interrupt Level 1, Line Printer Level 2, System Clock (factory-installed jumper) Level 5, iSBC 208 (factory-installed jumper) Level 6, Terminal Driver (Read) Level 7, Terminal Driver (Write) Clear-To-Send signal capability Set dual-port RAM address Enable Non-Maskable Interrupt Non-bus vector interrupt If iSBC 337 is <u>not</u> used. Selects 5MHz clock (required if iSBC 337 <u>is</u> used) Selects 2732-type EPROM EPROM address range Only for parallel priority resolution
	E119-E120 E232-E233	86/14 <u>with</u> iSBC 300A RAM expansion module
	E230-E231 E232-E233	86/14 <u>without</u> RAM expansion module
E44-E53 E45-E54 E46-E55 E48-E57 E50-E59 E51-E60 E52-E61	E44-E59 E45-E50 E45-E54 E46-E51 E48-E53 E50-E52	If parallel port is used for 957B
E44-E53 E51-E60	E60-E63	If parallel port is used for line printer

Table 5-21. iSBC® 86/14 Devices (Condensed)

Device	Part Number	Socket
2732A EPROM	144447-001	U57
2732A EPROM	144448-001	U58
2732A EPROM	144449-001	U39
2732A EPROM	144450-001	U40
902 Resistor Packs	4500645-01	U18, U20, U21
7438 IC	100908-001	U19 If parallel port used for Line Printer
Status Adapter	1002129	U19 If parallel port used for iSBC 957B

PREPARING YOUR HARDWARE

Table 5-22. iSBC® 86/30 Jumpers (Condensed)

Remove Jumper	Add Jumper	Function/Description
E144-E145 E219-E225 E26-E27 E33-E34 E111-E112 E210-E211	E165-E166 E145-E149 E132-E164 E147-E158 E151-E152 E153-E155 E134-E154 E76-E77 E61-E62 E36-E37 E124-E125 E112-E113	Interrupt Level 0, iSBC 337 Level 1, Non-Maskable Interrupt Level 1, Line Printer Level 2, System Clock (factory-installed jumper) Level 5, iSBC 208 (factory-installed jumper) Level 6, Terminal Driver (Read) Level 7, Terminal Driver (Write) Clear-To-Send signal capability Set dual-port RAM address Enable Non-Maskable Interrupt Non-bus vector interrupt If iSBC 337 is <u>not</u> used. Selects 5MHz clock (required with iSBC 337) Selects 2732-type EPROM EPROM address range Only for parallel priority resolution
	E119-E120	86/30 <u>with</u> iSBC 304 RAM expansion module
	E232-E233	86/30 <u>without</u> RAM expansion module
E44-E53 E45-E54 E46-E55 E48-E57 E50-E59 E51-E60 E52-E61	E44-E59 E45-E50 E45-E54 E46-E51 E48-E53 E50-E52	If parallel port is used for 957B
E44-E53 E51-E60	E60-E63	If parallel port is used for line printer

Table 5-23. iSBC® 86/30 Devices (Condensed)

Device	Part Number	Socket
2732A EPROM	144447-001	U57
2732A EPROM	144448-001	U58
2732A EPROM	144449-001	U39
2732A EPROM	144450-001	U40
902 Resistor Packs	4500645-01	U18, U20, U21
7438 IC	100908-001	U19 If parallel port used for Line Printer
Status Adapter	1002129	U19 If parallel port used for iSBC 957B

PREPARING YOUR HARDWARE

Table 5-24. iSBC® 208 Jumpers (Condensed)

Remove Jumper	Add Jumper	Function/Description
	E79-E84	Interrupt level 5
E45-E49	E41-E45	16-bit I/O address decoding.
E77-E78		Only if parallel bus priority resolution is used. (Factory sends board with this jumper removed, for serial bus priority resolution).

LINE PRINTER SIGNALS

Table 5-25. Line Printer Pin Assignments (Condensed)

50-Pin iSBC Connector	50-Pin Edge Connector	-- CENTRONICS-Standard -- 30-Pin Connector Signal	
24	23	1	Character strobe to printer
26	25	13	SLCT (Select)
28	27	12	Paper Out
30	29	10	ACKNOWLEDGE from printer
34	33	9	Data Bit 7
36	35	8	Data Bit 6
38	37	7	Data Bit 5
40	39	6	Data Bit 4
42	41	5	Data Bit 3
44	43	4	Data Bit 2
46	45	3	Data Bit 1
48	47	2	Data Bit 0
<p>GROUNDS:</p> <p>iSBC Connector: All odd pin numbers are grounded. Edge Connector: All even-numbered pins grounded. Centronics LP Connector: Pins 19-29 Protective grounds Pin 16 Logic Ground Pin 17 Chassis Ground</p>			

CHAPTER 6. DOCUMENTATION

This chapter lists and briefly describes documentation that applies to the iRMX 86 PC Operating System. We have included descriptions of iRMX 86 software manuals, as well as manuals describing hardware that can be used with the iRMX 86 PC product.

THIS MANUAL

The GETTING STARTED WITH THE iRMX 86 SYSTEM is designed to be a self-contained summary of the information you need to use the iRMX 86 PC Operating System. Much of the information in this manual is repeated in other manuals described here.

iRMX 86 MANUALS

These are the manuals that document the iRMX 86 Operating System.

- INTRODUCTION TO THE iRMX 86 OPERATING SYSTEM (Order Number: 9803124)

This manual is designed to introduce engineers and managers to the iRMX 86 Operating System. It describes how the iRMX 86 Operating System can help you develop your application system in less time and at less expense.

- iRMX 86 NUCLEUS REFERENCE MANUAL (Order Number: 9803122)

This manual documents the Nucleus, the central portion of the iRMX 86 Operating System required by all application systems. It provides overview information, discusses the functions of the Nucleus in detail, and contains detailed descriptions of the system calls available to application programmers.

- iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL (Order Number: 9803123)

This manual describes the Basic I/O System, a layer of the iRMX 86 Operating System that provides flexible I/O features that are useful in a broad range of applications. It contains some introductory and overview material as well as detailed descriptions of the system calls available to application programmers.

DOCUMENTATION

- iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL (Order Number: 143308)

This manual describes the Extended I/O System, a layer of the iRMX 86 Operating System that provides easy-to-use, more-automatic I/O features. It contains some introductory and overview material as well as detailed descriptions of the system calls available to application programmers.

- iRMX 86 LOADER REFERENCE MANUAL (Order Number: 143318)

This manual describes the two loaders available with the iRMX 86 Operating System: the Bootstrap Loader and the Application Loader. It contains some introductory and overview material as well as detailed descriptions of the system calls available with the Application Loader.

- iRMX 86 HUMAN INTERFACE REFERENCE MANUAL (Order Number: 9803202)

This manual documents the Human Interface, the layer of the iRMX 86 Operating System that provides an interactive interface between the user and the application system. It provides introductory and overview information, describes the commands available with the Human Interface (the same commands described in Chapter 3 of the manual you are reading), discusses the process of creating your own commands, and describes Human Interface system calls.

- iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL (Order Number: 144133)

This manual documents the Disk Verification Utility. The DISKVERIFY command (see Chapter 3 of the manual you are reading) invokes this utility. The DISK VERIFICATION UTILITY REFERENCE MANUAL provides more in-depth information, including detailed descriptions of the structure of iRMX 86 files.

- iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL (Order Number: 142721)

This manual documents advanced features of the iRMX 86 Operating System normally used by system programmers. The manual includes discussions of regions, attaching I/O devices, and creating user objects. It contains detailed descriptions of those system calls normally reserved for system programmers.

DOCUMENTATION

- iRMX 86 PROGRAMMING TECHNIQUES MANUAL (Order Number: 142982)

This manual provides a number of programming techniques that can reduce the amount of time you spend designing and implementing your iRMX 86-based application system. It includes discussions on PL/M-86 size controls, interface procedures, INCLUDE files, timer routines, assembly language programming, job communication, configuration, deadlock, terminal I/O, and stack sizes.

- GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O SYSTEMS (Order Number: 142926)

For the programmer who is using the configurable iRMX 86 Operating System, this manual shows how to incorporate a custom driver into the system. This applies to devices for which the iRMX 86 Operating System does not already supply device drivers.

- iRMX 86 CONFIGURATION GUIDE (Order Number: 9803126)

Again, for the programmer who is using the configurable iRMX 86 Operating System, this manual describes how to define the characteristics of iRMX 86 layers that are appropriate a particular application.

- iRMX 86 INSTALLATION GUIDE (Order Number: 9803125)

This manual contains hardware information for the configurable iRMX 86 Operating System (equivalent to the hardware information in this manual) and a description of the iRMX 86 Patching Utility.

DOCUMENTATION

LANGUAGE TRANSLATORS AND UTILITIES MANUALS

The following manuals document the language products that can be used with your iRMX 86 PC Operating System.

- EDIT REFERENCE MANUAL (Order Number: 143587)

This manual documents EDIT, an iRMX 86-based text editor. It contains introductory and tutorial material as well as detailed descriptions of all EDIT commands.

- GUIDE TO USING iRMX 86 LANGUAGES (Order Number: 143907)

This manual provides an overview of the language products that run in an iRMX 86 environment. It shows how to invoke the products from the Human Interface and lists the invocation controls for each product. It then refers you to other language and utilities manuals for detailed information about the products. You should read this manual before you read the other language and utilities manuals, because this manual provides information that you need to run the language products in an iRMX 86 environment. It also identifies portions of the other manuals that do not apply to the iRMX 86 versions of the language products.

- 8086/8087/8088 MACRO ASSEMBLY LANGUAGE REFERENCE MANUAL FOR 8086-BASED DEVELOPMENT SYSTEMS (Order Number: 121627)

This manual documents the 8086/8087/8088 macro assembly language, ASM86. It describes the assembly language instructions and the macro processing language.

- 8086/8087/8088 MACRO ASSEMBLER OPERATING INSTRUCTIONS FOR 8086-BASED DEVELOPMENT SYSTEMS (Order Number 121628)

This manual describes how to invoke the assembler, and how to link assembly language programs with PL/M-86 programs.

- PL/M-86 USER'S GUIDE FOR 8086-BASED DEVELOPMENT SYSTEMS (Order Number: 121636)

This manual describes the PL/M-86 language and use of the PL/M-86 compiler. It describes language statements, discusses compiler invocation, and documents each compiler control.

DOCUMENTATION

- iAPX 86,88 FAMILY UTILITIES USER'S GUIDE FOR 8086-BASED DEVELOPMENT SYSTEMS (Order Number: 121616)

This manual contains descriptions of the program development utilities:

- LINK86, which links 8086 object modules together and resolves global references between modules
- LOC86, which changes 8086 relocatable object modules into absolute modules
- LIB86, a utility that creates and maintains object libraries
- OH86, which converts 8086 absolute object modules to hexadecimal format

- Pascal-86 USER'S GUIDE (Order Number: 121539)

This manual describes the Pascal language and the use of the Pascal-86 compiler. It provides complete descriptions of all Pascal language statements, discusses compiler invocation, and documents each of the compiler controls. The Pascal-86 compiler is a strict implementation of the proposed ISO standard that also provides extensions of the language oriented toward microcomputers.

- FORTRAN-86 USER'S GUIDE (Order Number: 121570)

This manual describes the FORTRAN language and the use of the FORTRAN-86 compiler. It provides complete descriptions of all FORTRAN language statements, discusses compiler invocation, and documents each of the compiler controls. This FORTRAN-86 compiler produces code that is compatible with existing FORTRAN-86 code and includes many new features of the FORTRAN-77 standard.

- RUN-TIME SUPPORT MANUAL FOR iAPX 86, 88 APPLICATIONS (Order Number: 121776)

This manual describes the run-time aids that Intel offers for the iAPX 86, 88 family of processors. This is the basic reference manual for the Universal Development Interface used with Intel Operating Systems.

DOCUMENTATION

- USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE (Order Number 143979)

This manual provides general information, interfacing instructions, and programming information for the iSBC 957B loader and monitor. It provides detailed descriptions of the loader and monitor commands and describes how to connect an Intel development system to an iAPX 86-based boards. It also contains configuration information, which may be of little importance to you since the monitor is already configured and available in PROM as part of the iRMX 86 PC package.

HARDWARE MANUALS

These manuals document hardware that you can use with your iRMX 86 PC Operating System.

COMPUTERS

The computers that you can use for your iRMX 86 PC Operating System are described in these two manuals.

- iSBC 86/12A SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL (Order Number: 9803074) and
- iSBC 86/14 and iSBC 86/30 SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL (Order Number: 144044)

These two manuals describe, for each computer, principles of operation, programming considerations, and how to incorporate iSBC Multimodule units (like on-board RAM and the 8087 Numeric Processor Extension).

DISK CONTROLLER

- iSBX 208 FLEXIBLE DISK CONTROLLER HARDWARE REFERENCE MANUAL (Order Number: 143078)

The manual describes specifications, jumper configurations, programming considerations, and principles of operation of the iSBX 208 Flexible Disk Controller board.

DOCUMENTATION

MEMORY BOARDS

- iSBC 016A/032A/064A/028A/056A RAM MEMORY BOARD HARDWARE REFERENCE MANUAL (Order Number: 143572)

This manual describes specifications, jumper configurations, programming considerations, and principles of operation of iSBC 056A RAM memory boards.

CHASSIS/POWER SUPPLY

- iSBC 680/681 MULTISTORE USER SYSTEM PACKAGE HARDWARE REFERENCE MANUAL (Order Number: 162432)

This manual provides information about the iSBC 680-series module, which is a chassis containing a power supply and Multibus card cage in which you can install your Intel iSBC boards.

You can order any manual described in this chapter from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

APPENDIX A. iRMX™ 86 EXCEPTION CODES

This appendix contains the exception codes that are generated by the iRMX 86 Operating System. Exception codes are any condition codes other than E\$OK, the normal code. Exception codes are classed as either "Environmental Conditions" or "Programmer Errors", although the latter includes certain hardware errors.

The values of these exception codes fall into ranges based on the layer which first detects the condition. Table A-1 lists the layers and their respective ranges, with numeric values expressed in hexadecimal notation.

Table A-1. Exception Code Ranges

Layer	Environmental	Programming
Nucleus	0 to 1FH	8000 to 801FH
Basic I/O System	20 to 3FH	8020 to 803FH
Extended I/O System	40 to 5FH	8040 to 805FH
Application Loader	60 to 7FH	8060 to 807FH
Human Interface	80 to AFH	8080 to 80AFH
Universal Development Interface	C0 to DFH	80C0 to 80DFH
Reserved	130 to 14FH	8130 to 814FH

iRMX™ 86 EXCEPTION CODES

Table A-2 below shows the value of each code, the associated mnemonic, and a descriptive meaning. In addition, the table shows the the layer(s) of the system that could generate the code, in case you wish to refer the the appropriate manual. Since certain system calls in the iRMX 86 Operating System are considered to be System Programmer calls and are documented in the iRMX 86 System Programmer Reference Manual, this manual is also mentioned.

Table A-2. iRMX™ 86 Condition Codes

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
0H	E\$OK	* * * * *	No exceptional conditions (normal)
Environmental Conditions			
1H	E\$TIME	* * * * *	A time limit (possibly a limit of zero time) expired without a task's request being satisfied.
2H	E\$MEM	* * * * *	Insufficient available memory to satisfy a task's request.
3H	E\$BUSY	S	Another task currently has access to data protected by a region.
4H	E\$LIMIT	* * * * *	A task attempted an operation which, if it had been successful, would have violated a Nucleus-enforced limit.
5H	E\$CONTEXT	* * * * *	A system call was issued out of proper context.
6H	E\$EXIST	* * * * *	A token parameter has a value which is not the token of an existing object.
N	Nucleus Reference Manual	L	Loader Reference Manual
B	Basic I/O System Ref Manual	H	Human Interface Reference Manual
E	Extended I/O Sys Ref Manual	S	System Programmer's Ref Manual

iRMX™ 86 EXCEPTION CODES

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
7H	E\$STATE	*	A task attempted an operation which would have caused an impossible transition of a task's state.
8H	E\$NOT\$CON- FIGURED	* * * * *	This system call is not part of the present configuration.
9H	E\$INTER- RUPT\$SAT- URATION	*	An interrupt task has accumulated the maximum allowable amount of SIGNAL\$INTERRUPT requests.
0AH	E\$INTER- RUPT\$- OVERFLOW	*	An interrupt task has accumulated more than the maximum allowable amount of SIGNAL\$INTERRUPT requests.
20H	E\$FEXIST	* *	File already exists.
21H	E\$FNEXIST	* * * *	File does not exist.
22H	E\$DEVFD	* * *	Device and file driver are incompatible.
23H	E\$SUPPORT	* * * *	Combination of parameters not supported.
24H	E\$EMPTY\$- ENTRY	* *	The specified slot in a directory file is empty.
25H	E\$DIR\$END	* *	The specified slot is beyond the end of a directory file.
26H	E\$FACCESS	* * * *	File access not granted.
27H	E\$FTYPE	* * *	Incompatible file type.
28H	E\$SHARE	* * * *	Improper file sharing requested.
29H	E\$SPACE	* *	No space left.
N	Nucleus Reference Manual	L	Loader Reference Manual
B	Basic I/O System Ref Manual	H	Human Interface Reference Manual
E	Extended I/O Sys Ref Manual	S	System Programmer's Ref Manual

iRMX™ 86 EXCEPTION CODES

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
2AH	E\$IDDR	* *	Invalid device driver request.
2BH	E\$IO	* * * *	An I/O error occurred.
2CH	E\$FLUSHING	* * * *	Connection specified in call was deleted before the operation was completed.
2DH	E\$ILLVOL	S *	Invalidly named volume.
2EH	E\$DEV\$OFF-LINE	*	The device being accessed is now offline.
40H	E\$PREFIX\$-SYNTAX	* *	The specified path starts with a colon (:), but does not contain a second, matching colon.
41H	E\$CANNOT\$-CLOSE	*	The Extended I/O System was not able to transfer remaining data in buffers to output device.
42H	E\$IOMEM	* *	The Basic I/O System has insufficient memory to process a request.
44H	E\$MEDIA	* *	The device containing a specified file is not online.
45H	E\$LOG\$NAME-NEXIST	* *	The Extended I/O System was unable to find a specified logical name in the object directories that it checks.
60H	E\$ABS\$ADDRESS	*	An absolute object program was loaded into system protected memory area.
61H	E\$BAD\$GROUP	* *	Illegal group component in the a group definition record.
62H	E\$BAD\$-HEADER	* *	Illegal header record in the object file.
N	Nucleus Reference Manual		L Loader Reference Manual
B	Basic I/O System Ref Manual		H Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		S System Programmer's Ref Manual

IRMX™ 86 EXCEPTION CODES

Table A-2. IRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
63H	E\$BAD\$SEG- MENT	* *	Illegal segment definition record.
64H	E\$CHECKSUM	* *	A checksum error occurred while reading an object record.
65H	E\$EOF	* *	Unexpected end of file encountered while reading object records.
66H	E\$FIXUP	* *	Illegal fixup record in the object file.
67H	E\$NO\$LOADER \$MEM	* *	Insufficient memory to satisfy loader dynamic memory requirements.
68H	E\$NO\$MEM	* *	Insufficient memory to create PIC/LTL segments.
69H	E\$REC\$FMT	* *	Illegal record format encountered.
6AH	E\$REC\$- LENGTH	* *	Record length of an object record exceeds configured loader-buffer size.
6BH	E\$REC\$TYPE	* *	Illegal record type encountered in the object file.
6CH	E\$NO\$START	* *	Start address not found.
6DH	E\$JOB\$SIZE	* *	Maximum job-size specified is less than the memory requirement specified in the object file.
6EH	E\$OVLY	*	Overlay name does not match with any of the overlay module names.
6FH	E\$LOADER \$SUPPORT	* *	The object file being loaded requires features not supported by the configured loader.
N	Nucleus Reference Manual	L	Loader Reference Manual
B	Basic I/O System Ref Manual	H	Human Interface Reference Manual
E	Extended I/O Sys Ref Manual	S	System Programmer's Ref Manual

iRMX™ 86 EXCEPTION CODES

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Environmental Conditions (continued)			
80H	E\$LITERAL	*	The parse buffer contains a literal with no closing quote.
81H	E\$STRING\$- BUFFER	*	The string to be returned as the parameter name exceeds the size of the buffer the user provided in the call.
82H	E\$SEPARA- TOR	*	The parse buffer contains a command separator.
83	E\$CONTINUED	*	The parse buffer contains a continuation character.
85H	E\$LIST	*	The last value of the value list is missing.
87H	E\$PREPOSI- TION	*	The same preposition as on the the command line was indicated, but can not be used.
89H	E\$CONTROL\$C	*	The user typed CONTROL-C while the command was being loaded.
8BH	E\$EXTRA\$SO	*	There were no more input pathnames although the output pathname list was not empty.
Programmer Errors			
8000H	E\$ZERO\$- DIVIDE	*	A task attempted to divide by zero.
8001H	E\$OVER-FLOW	*	An overflow interrupt occurred.
8002H	E\$TYPE	* * * * *	A token parameter referred to an existing object that is not of the required type.
N	Nucleus Reference Manual		L Loader Reference Manual
B	Basic I/O System Ref Manual		H Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		S System Programmer's Ref Manual

iRMX™ 86 EXCEPTION CODES

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Programmer Errors (continued)			
8003H	E\$BOUNDS	*	A task attempted to access beyond the end of a segment.
8004H	E\$PARAM	* * * * *	A parameter which is neither a token nor an offset has an invalid value.
8006H	E\$ARRAY\$- BOUNDS	* * *	Array overflow detected by hardware or language processor.
8007H	E\$NDP\$- ERROR	*	8087 (Numeric Data Processor) error.
8020H	E\$IFDR	* *	Invalid file driver request.
8021H	E\$NOUSER	* * *	No default user.
8022H	E\$NO\$PREFIX	* * *	No default prefix.
8040H	E\$NOT\$- PREFIX	* *	Specified object is not a device connection or file connection.
8041H	E\$NOT\$- DEVICE	*	A token parameter referred to an existing object that is not, but should be, a device connection.
8042H	E\$NOT\$CON- NECTION	*	A token parameter referred to an existing object that is not, but should be, a file connection.
8060H	E\$JOB\$PARAM	* *	The maximum job-size specified is less than the minimum job-size.
8080H	E\$PARSE\$- TABLES	*	There is an error in the internal parse tables.
8083H	E\$DEFAULT\$SO	*	The default output name STRING is invalid.
8084H	E\$STRING	*	The pathname to be returned exceeds 255 characters in length.
N	Nucleus Reference Manual		L Loader Reference Manual
B	Basic I/O System Ref Manual		H Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		S System Programmer's Ref Manual



APPENDIX B. iRMX™ 86 SYSTEM CALLS

This chapter describes the system calls that the iRMX 86 Operating System recognizes. If you wish to use any of these calls with the iRMX 86 PC System, you must obtain the manual that describes the system call (manuals are listed in Chapter 6) and you must link your programs to the appropriate library on the Library Diskette supplied with the iRMX 86 PC System. This Appendix lists the iRMX 86 System calls and briefly describes each call. The first section describes each layer of the Operating System.

LAYERS OF THE iRMX 86 SYSTEM

The Configurable iRMX 86 Operating System consists of a number of layers. The Operating System can be configured to include or exclude certain layers (the Nucleus is always included) and to include or exclude optional features. (The configuration process has already been accomplished for users of the iRMX 86 PC Operating System.)

The layers of the iRMX 86 Operating System are:

- | | |
|---------------------|---|
| Nucleus | The Nucleus is the core of the iRMX 86 Operating System and is required by every application system. It provides facilities that perform processor management and scheduling, interrupt management, memory management, object control, and error management. Refer to the iRMX 86 NUCLEUS REFERENCE MANUAL and the iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL for detailed information about the Nucleus. |
| Basic I/O System | The Basic I/O System provides an extensive facility for device-independent I/O. It supplies all file drivers and a number of device drivers. It implements an asynchronous interface to I/O operations, allowing tasks explicitly to overlap I/O functions with other operations. Refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL and the iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL for more information. |
| Extended I/O System | The Extended I/O System provides a higher-level interface to files than the Basic I/O System provides. The Extended I/O System provides a simple, synchronous interface to I/O operations, one which automatically performs read-ahead and write-behind buffering. This synchronous interface also allows tasks to use logical names to refer to files. All of the UDI File Management system calls (see Chapter 4 of this manual) are accomplished by the Extended I/O System. |

IRMX™ 86 SYSTEM CALLS

Refer to the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL and the iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL for more information.

Application Loader The Application Loader provides a simple mechanism for loading application code and data files from I/O devices into system memory. It can load absolute code into fixed locations, relocatable code into dynamically-allocated memory locations, and it can load files containing overlays.

Bootstrap Loader The Bootstrap Loader provides a means of loading the Operating System into system memory from an I/O device. It can also load a file you specified at the terminal. The Bootstrap Loader is in the EPROMs supplied with your iRMX 86 PC System.

Human Interface The Human Interface is the uppermost layer of the iRMX 86 Operating System. It is an interactive interface between you and the application system. Using the Human Interface, you can invoke a program from the terminal by specifying the name of the file that contains the program. A set of programs, the Human Interface Commands, are supplied with the Operating System. These are the commands documented in Chapter 3 of this manual.

The Human Interface also provides a number of system calls that the application program can invoke to access Human Interface services. Refer to the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL for more information.

NUCLEUS SYSTEM CALLS

The Nucleus system calls are listed here.

ACCEPT\$CONTROL	Gains control of a region only if the region is immediately available.
CATALOG\$OBJECT	Enters a name and token for an object into the object directory of a job.
CREATE\$JOB	Creates an environment for a number of tasks and other objects, as well as creating an initial task and its stack.
CREATE\$MAILBOX	Creates a mailbox with queues for waiting tasks and objects with FIFO or PRIORITY dicipline.
CREATE\$SEGMENT	Dynamically allocates a specified number of 16-byte paragraphs.
CREATE\$SEMAPHORE	Creates a semaphore for synchronizing access to resources.
CREATE\$TASK	Creates a task with the specified priority and stack area.
DELETE\$JOB	Deletes a Job and all the objects currently defined within its bounds only if that Job does itself not contain any other jobs. All memory used is returned to the containing job.
DELETE\$MAILBOX	Deletes a mailbox.
DELETE\$SEGMENT	Deletes the specified segment by deallocating the memory.
DELETE\$SEMAPHORE	Deletes a semaphore.
DELETE\$TASK	Deletes a task from the system, and removes it from any queues in which it may be waiting.
DISABLE	Disables the hardware from accepting interrupts at or below a specified level.
ENABLE	Enables the hardware to accept interrupts from a specified level.
EXIT\$INTERRUPT	Used by an interrupt handler to relinquish control of the System.
GET\$LEVEL	Returns the number of the highest priority interrupt level currently being processed.
GET\$POOL\$ATTRIBUTES	Returns attributes such as the minimum and maximum, as well as current size of the memory in the environment of the calling task's job.

iRMX™ 86 SYSTEM CALLS

GET\$PRIORITY	Obtains the current priority of a specified task.
GET\$SIZE	Returns the size (in bytes) of a segment.
GET\$TASK\$TOKENS	Gets the token for the calling task or associated objects within its environment.
GET\$TYPE	Returns a code for the type of object referred to by the specified token.
LOOKUP\$OBJECT	Returns a token for the object with the specified name found in the object directory of the specified job.
OFFSPRING	Provides a list of all the current Jobs created by the specified job.
RECEIVE\$MESSAGE	Attempts to receive an object from a specified mailbox. The calling task may choose to wait for a specified number of system time units if no object is available.
RECEIVE\$UNITS	Attempts to gain a specified number of units from a semaphore. If the units are not immediately available, the calling task may choose to wait.
RESET\$INTERRUPT	Disables an interrupt level, and cancels the assignment of the interrupt handler for that level. If an interrupt task was assigned, it is deleted.
RESUME\$TASK	Resumes a task. If the task had been suspended multiple times, the suspension depth is reduced by one, and it remains suspended.
SEND\$CONTROL	Relinquishes control of a region.
SEND\$MESSAGE	Sends an object to a specified mailbox. If a task is waiting, the object is passed to the appropriate task according to the queuing discipline. If no task is waiting, the object is queued at the mailbox.
SEND\$UNITS	Increases a semaphore counter by the specified number of units.
SET\$INTERRUPT	Assigns an interrupt handler and, if desired, an interrupt task to the specified interrupt level. Usually the calling task becomes the interrupt task.
SET\$POOL\$MIN	Dynamically changes the minimum memory requirements of the job environment containing the calling task.

iRMX™ 86 SYSTEM CALLS

SET\$PRIORITY	Dynamically alters the priority of the specified task.
SIGNAL\$INTERRUPT	Used by an interrupt handler to signal the associated interrupt task that an interrupt has occurred.
SLEEP	Causes a task to enter the ASLEEP state for a specified number of system time units.
SUSPEND\$TASK	Suspends the operation of a task. If the task is already suspended, its suspension depth is increased by one.
UNCATALOG\$OBJECT	Removes an object and its name from a job's object directory.
WAIT\$INTERRUPT	Used by an interrupt task to SLEEP until the associated interrupt handler signals the occurrence of an interrupt.

BASIC I/O SYSTEM CALLS

These are the Basic I/O System calls.

A\$ATTACH\$FILE	Creates a connection to an existing file and returns its token identifier.
A\$CHANGE\$ACCESS	Changes the types of accesses permitted to the specified user(s) for a specific file.
A\$CLOSE	Closes the connection to the specified file so that it may be used again, or so that the type of access may be changed.
A\$CREATE\$DIRECTORY	Creates a Named File used to store the names and locations of other named files, and returns a token identifier for the connection to the new file.
A\$CREATE\$FILE	Creates a data file with the specified access rights, and returns a token identifier for the connection to the new file.
A\$DELETE\$CONNECTION	Deletes the connection to the specified file.
A\$GET\$FILE\$STATUS	Returns the current status of a specified file.
A\$OPEN	Opens a file for either read, write, or update access.
A\$READ	Reads a number of bytes from the current position in a specified file.

iRMX™ 86 SYSTEM CALLS

A\$SEEK	Moves the current data pointer of a named or physical file.
A\$WRITE	Writes a number of bytes at the current position in a file.

EXTENDED I/O SYSTEM CALLS

These are the Extended I/O System calls.

CREATE\$I/O\$JOB	Creates an I/O job with one task.
EXIT\$I/O\$JOB	Sends a message to a previously designated mailbox and deletes the calling task.
S\$ATTACH\$FILE	Creates a connection to an existing file.
S\$CATALOG\$CONNECTION	Creates a logical name for a connection by cataloging the connection in the object directory of a specific job.
S\$CHANGE\$ACCESS	Changes the access list for a named file.
S\$CLOSE	Closes an open connection to a named, physical or stream file.
S\$CREATE\$DIRECTORY	Creates a new directory file.
S\$CREATE\$FILE	Creates a new physical, stream, or named data file. It cannot create a named directory file.
S\$CREATE\$I/O\$JOB	Creates an I/O job containing one task.
S\$DELETE\$CONNECTION	Deletes a file connection. It cannot delete a device connection.
S\$DELETE\$FILE	Deletes a stream, physical, or named file.
S\$EXIT\$I/O\$JOB	Sends a message to a previously designated mailbox and deletes the calling task.
S\$GET\$CONNECTION\$STATUS	Provides status information about file and device connections.
S\$GET\$FILE\$STATUS	Allows a task to obtain information about a physical, stream, or named file.
S\$LOOK\$UP\$CONNECTION	Searches through an I/O job's local, global, and root object directories to find the connection associated with a logical name.

iRMX™ 86 SYSTEM CALLS

S\$OPEN	Opens a connection to a named, physical, or stream file.
S\$READ\$MOVE	Reads a number of bytes from a file to a buffer.
S\$RENAME\$FILE	Changes the path of a named file. It cannot be used for stream or physical files.
S\$SEEK	Moves the file pointer.
S\$SPECIAL	Allows your task to perform functions that are peculiar to a specific device.
S\$TRUNCATE\$FILE	Removes information from the end of a named data file.
S\$UNCATALOG\$CONNECTION	Deletes a logical name from the object directory of a specific job.
S\$WRITE\$MOVE	Writes a collection of bytes from a buffer to a file.

HUMAN INTERFACE SYSTEM CALLS

These are the Human Interface System Calls.

C\$CREATE\$COMMAND\$CONNECTION	Create a command connection and return a token.
C\$DELETE\$COMMAND\$CONNECTION	Delete a specific command connection.
C\$FORMAT\$EXCEPTION	Format a default message into a user buffer for a given exception code.
C\$GET\$CHAR	Get a character from the command line.
C\$GET\$INPUT\$CONNECTION	Return an EIOS connection for the specified input file.
C\$GET\$INPUT\$PATHNAME	Parse the command line return a pathname that will identify the Standard Input file.
C\$GET\$OUTPUT\$CONNECTION	Return an EIOS connection for the specified output file.
C\$GET\$OUTPUT\$PATHNAME	Parse the command line and return a pathname that will identify the Standard Output file.

iRMX™ 86 SYSTEM CALLS

C\$GET\$PARAMETER	Parse the command line for the next parameter and return it as a keyword name and a value.
C\$SEND\$CO\$RESPONSE	Send a message to the command output (CO) and read a response from the command input (CI).
C\$SEND\$COMMAND	Concatenate command lines into the data structure created by CREATE\$COM- MAND\$CONNECTION and then execute command.
C\$SEND\$EO\$RESPONSE	Send a message to the error output device (EO) and return a response from the error input device (EI).
C\$SET\$CONTROL\$C	Change calling program's CONTROL C semaphore to the specified semaphore.
C\$SET\$PARSE\$BUFFER	Parse a buffer other than the current command line.

SYSTEM PROGRAMMER SYSTEM CALLS

These system calls are considered System Programmer calls because of their global effect on the system.

A\$GET\$EXTENSION\$DATA	Returns from the I/O System extension data stored with a file.
A\$PHYSICAL\$ATTACH\$DEVICE	Attaches a device to the Basic I/O System.
A\$PHYSICAL\$DETACH\$DEVICE	Detaches a device from the Basic I/O System.
A\$SET\$EXTENSION\$DATA	Sets the extension data for a file from the I/O System.
ACCEPT\$CONTROL	Requests access to data protected by a region only if access is immediately available.
ALTER\$COMPOSITE	Alters the component list of a composite object.
CREATE\$COMPOSITE	Creates a composite object.
CREATE\$EXTENSION	Creates a new extension object type.
CREATE\$REGION	Creates a region.
CREATE\$USER	Creates a user object.

iRMX™ 86 SYSTEM CALLS

DELETE\$COMPOSITE	Deletes a composite object.
DELETE\$EXTENSION	Deletes an extension type.
DELETE\$REGION	Deletes a region.
DELETE\$USER	Deletes a specified user object.
DISABLE\$DELETION	Increases the deletion disabling depth of an object by one.
ENABLE\$DELETION	Decreases the deletion disabling depth of an object by one.
FORCE\$DELETE	Forces the deletion of an object even if the object has had its deletion disabled once.
INSPECT\$COMPOSITE	Returns a list of the component object tokens contained in a composite object.
INSPECT\$USER	Returns a list of the ID's in a user object.
LOGICAL\$ATTACH\$DEVICE	Attaches a device to the Extended I/O System.
LOGICAL\$DETACH\$DEVICE	Detaches a device from the Extended I/O System.
RECEIVE\$CONTROL	Requests eventual access to data protected by a region.
SEND\$CONTROL	Relinquishes access to data protected by a region.
SET\$OS\$EXTENSION	Allocates and deallocates extension entries in the interrupt vector table.
SET\$PRIORITY	Changes the priority of a task dynamically.
SET\$TIME	Sets the time and the date.
SIGNAL\$EXCEPTION	Signals the occurrence of an exceptional condition.

APPENDIX C. MONITOR COMMANDS

The iRMX 86 PC Operating System includes the iSBC 957B Monitor, which resides in EPROM on the processor board. This appendix describes the Monitor commands, which allow you to do such things as:

- Set breakpoints in programs
- Single-step through your programs
- Examine and modify registers and memory
- Perform I/O via 8086 input and output ports
- Move and compare blocks of memory

Also, by connecting your hardware to an Intel Microcomputer Development System (using the iSBC 957B package), you can use the monitor from the Development System. Chapter 5, PREPARING YOUR HARDWARE, describes the jumpering and devices required on your Single Board Computer to support this feature. The USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE describes the iSBC 957B package.

You can get to the monitor in any of these ways:

- By booting the system (as described in Chapter 2), and when the period (.) prompt is displayed, the Monitor is ready to accept commands.
- By using the Human Interface DEBUG command, specifying a program file. This loads a program into memory and gives control to the monitor, permitting you to examine the program in detail. The DEBUG command is described in Chapter 3.
- By pressing a button connected to the nonmaskable interrupt of your Single Board Computer. This interrupts the application system and gives control to the monitor, which prompts with a period (.) and waits for your entry.

CAUTION

To prevent destroying data on your diskettes, wait at least 2 seconds after your last iRMX 86 command before you interrupt the computer.

In this chapter, the 8087 Numeric Processor Extension is referred to as "NPX."

MONITOR COMMANDS

COMMAND STRUCTURE

Responses to the monitor's command-level prompt are line-oriented, as opposed to the more traditional character-oriented monitor input. This allows for command-line editing capabilities.

Each monitor command includes a key letter, which is suggestive of the function of the command, such as D for displaying memory and S for substituting memory. Some commands have one or more additional letters which specify variations of the general function.

Following the key letter or letters of a command are zero or more arguments. The arguments can be addresses, data, register names, strings, or punctuation symbols depending on the command.

In the remainder of this manual, the following syntax conventions are used:

[A]	indicates that "A" is optional
[A]*	indicates zero or more optional iterations of "A"
	indicates that "B" is a variable
{A B}	indicates "A" or "B"
<cr>	indicates a carriage return

Variables in commands include numbers, registers, expressions, and addresses. The BYTE and WORD variables are defined in the following sections.

BYTE AND WORD VARIABLES

```
<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> ::= {<hex number>|<dec number>T}
<register> ::= {AX|BX|CX|DX|SP|BP|SI|DI|CS|DS|SS|ES|IP|FL}
<term> ::= {<number>|<register>}
<expr> ::= {<term>|<expr>{+|-}<term>}
<addr> ::= {[<expr>:]<expr>}
<range> ::= {<addr>|<addr>#<number>}
```

The range of byte values is 00-OFFH. Larger numbers can be entered but only the last two digits are significant because the number is evaluated modulo 256. The range of word values is 0000-OFFFFH. Larger numbers can be entered, but only the last four hex digits are significant because the number is evaluated modulo 65536. Leading zeros can be omitted for both types of values.

Byte and word values are assumed to be in hexadecimal. However, decimal values can be entered if they end with a "T". The trailing "H" that sometimes indicates hexadecimal is not allowed for byte or word values.

MONITOR COMMANDS

When word values are displayed, the contents of the high byte of the address location is displayed, followed by the contents of the low byte of the address location. Similarly, when entering word values, the high byte is followed by the low byte. If necessary, leading zeros are appended to the value by the monitor. Assume, for example, that the byte values C4, 26, F2, and 3D are in consecutive locations beginning at 246B:26. A display of those locations in bytes looks like:

```
246B:0026 C4 26 F2 3D
```

while the corresponding display in words looks like:

```
246B:0026 26C4 3DF2
```

NUMERIC (REAL, INTEGER AND BCD) VARIABLES

```
<sign> ::= [+|-]
<npx dec number> ::= <sign><dec number>
<npx hex number> ::= <hex number>H
<scientific number> ::= {<npx dec number>[.<dec number>]|
                           <sign>.<dec number>}[E<npx dec number>]
<int number> ::= {<npx dec number>|<npx hex number>}
<BCD number> ::= {<npx dec number>|<npx hex number>}
<real number> ::= {<scientific number>|<npx dec number>|<hex number>R}
<npx register> ::= {CW|SW|TW|OP|DP}
<npx stack register> ::= ST[({0|1|2|3|4|5|6|7})]
```

Numeric variables refer to the data types supported by the 8087 Numeric Processor Extension (NPX). There are three types of numeric variables: integer, packed binary coded decimal (BCD), and real. Of these three basic types, the integer and real types have three sub-types. All seven numeric data types are described in Table C-1. For the remainder of this manual, the seven numeric variables are referred to as "NPX data types."

See the 8086 FAMILY USER'S MANUAL NUMERICS SUPPLEMENT for more details on the NPX data types. Also, note the section on "Constants" in the 8086/8087/8088 MACRO ASSEMBLY LANGUAGE REFERENCE MANUAL. For other NPX related details, refer to the Application Note, Getting Started With the Numeric Data Processor.

MONITOR COMMANDS

Table C-1. NPX Data Types

Data Type	Explicit Suffix	Bits	Significant Digits (Decimal)	Approximate Range (Decimal)
Word integer	H	16	4	$-32,768 < X < +32,767$
Short integer	H	32	10	$-2 \times 10^9 < X < +2 \times 10^9$
Long integer	H	64	19	$-9 \times 10^{18} < X < +9 \times 10^{18}$
Packed decimal	H	80	18	$-99..99 < X < +99..99(18 \text{ digits})$
Short real*	R	32	6-7	$8.43 \times 10^{-37} < X < 3.37 \times 10^{38}$
Long real*	R	64	15-16	$4.19 \times 10^{-307} < X < 1.67 \times 10^{308}$
Temporary real	R	80	19	$3.4 \times 10^{-4929} < X < 1.2 \times 10^{4929}$
<p>* The short and long real data types correspond to the single and double precision data types defined in other Intel numeric products.</p>				

The suffixes used when entering the NPX data types differ from the suffixes for word and byte variables. If the no suffix is given when entering an NPX data type, the number is assumed to be a decimal number. A decimal number is defined for the real NPX data types as a value entered as a scientific number. This allows values like 4, 1.2, -1.2, -.3, -.3E-44, -1.56E-999 or 5.67E55 to be entered. A decimal number is defined for the integer and BCD NPX data types as a value entered as a scientific number that will evaluate to an integer value. This allows numbers like 12, -12, 4E2 or 4.0E1 to be entered but won't allow the entry of numbers like 1.2, -1.2 or -1.56E-999. In the valid cases, the monitor will place the hexadecimal equivalent of the input decimal number into iAPX 86, 88 memory. However, if an integer or BCD number is entered with its explicit suffix "H" or a real number is entered with its explicit suffix "R", the monitor places the number, as it is entered at the console, into iAPX 86, 88 memory. In this case explicit signs (+ or -) are not allowed, the hexadecimal number, entered at the console, indicates the sign of the number in the sign bit, the most significant bit.

MONITOR COMMANDS

When NPX data types are displayed, the address of the data type is displayed and then the value is displayed in hexadecimal form. The number is then displayed as the equivalent decimal number if it has an equivalent decimal value. For example, the long real number 11223344, is displayed in form:

```
1111:0 4165682600000000R    11223344
```

The long integer, 11223344, is displayed in the form:

```
1111:0 0000000000AB4130H    11223344
```

The BCD number, 11223344, is displayed in the form:

```
1111:0 00000000000011223344T 11223344
```

In the remainder of this manual this display form is referred to as "NPX number format". If the memory value is a special bit pattern identifying non-numeric values like Not-A-Number (NAN) or Infinity, the address and the hexadecimal number are displayed and then the meaning is shown as NAN or Infinity instead of the decimal value. Examples of these displays using a long real number are:

```
0080:0000 FFFF000000000000R  -NAN
0080:0008 7FF0000000000000R  +Infinity
```

Special cases of numeric values are also identified. A negative zero is displayed as -0. Pseudo zeroes (zero fraction with non-zero exponent) are shown as 0Eexp, where exp is the base 10 power equivalent of the binary exponent in the number. Numbers which are not normalized (I bit is zero) are displayed with their hexadecimal value and a "Bit" value which is a count of how many leading zeroes existed in the number. This "Bit" value indicates how many times the fractional part of the number must be shifted to the left to normalize it. An example of this display using a temporary real number is:

```
0080:000 3FFF19999999999999AR  .2 UNNORM 3 BITS
```

Decimal values can be displayed in any of four different formats. The format used depends on the range of the number and its value. Numbers which are exact integers and fit in the field size of 16 digits are displayed as integers with no trailing decimal point or 0. An example of this display using a long real number is:

```
0080:0000 43118B54F22AEBOOR  1234567890123456
```

Values which appear as integers, within the limits of the field size, but are not exact integers are displayed as XXXXX.0. The .0 suffix indicates that the value is close to an integer but not exactly. An example of this display using a long real number is:

```
0080:0000 42DC12218377DE46R  123456789012345.0
```

MONITOR COMMANDS

If the magnitude of a number is greater than or equal to 0.1 and is less than $10^{**}<\text{field size}>$, the number is displayed as XXXX.XXX. An example of this display using a long real number is:

```
0080:0000 41D26480B487E69BR 1234567890.12345
```

Finally, very large or very small numbers are displayed in scientific number format X.XXXXXEexp. An example of this display using a long real number is:

```
0080:0000 492C2916217B84B7R 3.14E+44
```

Trailing zeroes after the decimal point are also suppressed.

When NPX data types are displayed, the most-significant byte of the memory address (in hexadecimal notation) is displayed in the leftmost position, followed by bytes of decreasing significance with the least significant byte in the right most position. Similarly, when entering NPX data types in hexadecimal or decimal, the first digit entered has the greatest significance and successive digits entered have decreasing significance. If less than the NPX data type's number of significant digits is entered, the monitor will append leading zeros. When entering a value for an NPX data type in scientific number format, the number is converted to its hexadecimal equivalent and is then stored in iAPX 86, 88 memory in that format.

ADDRESS SPECIFICATION

A complete address argument consists of a base and an offset separated by a colon (:). If the optional base portion is omitted, the contents of the iAPX 86, 88 CS register are used as a default base, except as noted in the command descriptions that follow. If an entire address is omitted, but an address is needed in the command, the contents of the CS and IP registers are used, respectively, as base and offset, except as noted in the command description.

There are two ways of denoting a range of addresses. One way is to list both the starting and ending addresses, with an exclamation point between them. An example is 30:46D ! 30:4FE. The other way is to list the starting address and the length in bytes, with a pound sign (#) between them. An example equivalent to the earlier one is 30:46D # 92.

If the ending address in a range lacks an explicit base part, the base of the starting address is assumed. The ending address may not contain a base part which differs from the base part of the starting address.

The largest count or the maximum number of bytes specified by a range is 0FFFFh. When a range is expected and neither an ending address nor a length is specified, the range is taken to be a single byte.

MONITOR COMMANDS

MULTIPLE COMMANDS ON A SINGLE LINE

There are two mechanisms for putting more than one command on a command line. First, separate commands may be in the same command line if they are separated by semicolons (;). Second, by enclosing a command in angle brackets (<command>) and by placing a decimal repetition factor ahead of the first bracket, you can specify that the command be repeated the desired number of times. A repetition factor of n says "do this command n times." For example,

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

is a valid command line that is built from the commands G, CS:3B7 and D DS:4A. The command G, CS:3B7 is repeated 12 times, then the D DS:4A is performed once. This entire sequence is repeated 5 times so the G, CS:3B7 command is repeated at total of 60 times while the D DS:4A command is repeated a total of only 5 times. Note that this use of angle brackets is NOT the same as the use of angle brackets in the syntax definition.

Closely related to repetition, but differing, is continuation. By putting a decimal continuation factor, n, immediately ahead of a command's key letter or letters, you are directing the monitor to "do this command for n items at a time." For example, the command D 200:14 directs the monitor to display the byte at address 200:14, while 20D 200:14 causes the display of 20 consecutive bytes, beginning at address 200:14. In contrast, 20 <D 200:14> causes the byte at 200:14 to be displayed 20 times.

NOTE

Both repetition and continuation factors are written as positive decimal integers with no "T" suffix. The range of these factors is 1 through 65,535. In any other part of a command using byte or word variables, however, decimal integers must have a "T" suffix, such as 127T.

MONITOR COMMANDS

iAPX 86 AND iAPX 88 CPU REGISTERS

The iAPX 86 and iAPX 88 CPUs include the 14 registers listed in Table C-2. The abbreviations used in the table are those used in the command syntax.

Table C-2. iAPX 86, 88 CPU Registers

Register Name	Abbreviation
Accumulator	AX
Base	BX
Count	CX
Data	DX
Stack Pointer	SP
Base Pointer	BP
Source Index	SI
Destination Index	DI
Code Segment	CS
Data Segment	DS
Stack Segment	SS
Extra Segment	ES
Instruction Pointer	IP
Flag	FL

NPX REGISTERS

The NPX includes the eight 80-bit individually-addressable stack registers plus the status word, control word, tag word, instruction pointer, data pointer and instruction opcode field listed in Table C-3. The abbreviations used in the table are those used in the command syntax. Note that the NPX instruction pointer listed in Table C-3 is not the iAPX 86, 88 instruction pointer. The monitor contains no command to modify the NPX instruction pointer.

MONITOR COMMANDS

Table C-3. NPX Registers

Register Name	Abbreviation
NPX State	N
Status Word	SW
Control Word	CW
Tag Word	TW
Instruction Pointer	IP
Data Pointer	DP
Instruction Opcode	OP
Stack Register 0	St(0)
.	.
.	.
.	.
Stack Register 7	St(7)

ERRORS

Each line input to the monitor is checked for validity. If the command is invalid or impossible to execute, an explanatory error message is displayed. If the command line containing the error consists of multiple commands, any valid commands prior to the error are executed.

Three error messages - "Bad EMDS Connection", "Bad Patch Byte=hex number", and XISIS Abort" - are all indicative of hardware problems. To recover, check your hardware, restart monitor, and try again.

ENTERING COMMANDS

The monitor's command line editor responds to input as follows:

- Digits, upper and lower case letters, and all other standard keyboard characters are accepted into the command line and are printed on the console. Upper and lower case letters are indistinguishable to the monitor, all display is done by the monitor in upper case.
- RUBOUT deletes the most recently entered character (with backspace, space, backspace) from both the command line and the display. An attempt to rubout the prompt causes a beep to be sounded.

MONITOR COMMANDS

- CNTRL/C directs the monitor to abort its current command and issue a prompt. However, if your program is running and is in a loop, CNTRL/C has no effect.
- CNTRL/R displays at the console the current command line. If the console terminal is in transparent mode, however, control-R has no effect.
- CNTRL/X deletes the current command line and displays a pound sign (#).
- CNTRL/S causes the console output to be suspended at the current cursor position. No output is lost by this command.
- CNTRL/Q causes the console output, suspended by Control-S, to be resumed beginning at the current cursor position.
- CARRIAGE RETURN (CR) signals the completion of the command line, which is then read and acted upon.
- Other characters have no effect. Spaces may be included anywhere in the command line except within lexical elements.
- NPX data types may be entered only on the substitute ("S") or "XST(n)" command line and may appear in no other command line.

Command lines may be up to 255 characters in length. An attempt to exceed this limit will be unsuccessful and will cause the terminal to beep.

MONITOR COMMANDS

COMMAND DESCRIPTIONS

The Monitor commands are summarized in Table C-4.

Table C-4. Summary of Loader And Monitor Commands

COMMAND	FUNCTION AND SYNTAX
L Load	Loads an absolute object file from Intellec into iAPX 86, 88 memory. L <filename> <cr>
G Go	Transfers control of the CPU to the user program. G [<start-addr>][, <break-addr> <range>]<cr>
R Load and Go	Loads an absolute object file from Intellec into iAPX 86, 88 memory and begins execution. R<filename><cr>
T Upload	Loads a block of iAPX 86, 88 memory into an Intellec file. T<range> , <filename> [, <start-addr>]<cr>
N Single Step	Displays and executes one instruction at a time. [<cont>] N [O] [P] [Q] [<start-addr>][,]<cr>
X Examine	Displays or modifies iAPX 86, 88 or NPX registers. X[<reg>[=<expr>]]<cr> X{N [<npx register>[=<hex number>]] [<npx stack register>[=<real number>]]}<cr>
D Display	Displays contents of a memory block. [<cont>] D [{W I SI LI T SR LR TR X}] [<range>][,]<cr>
S Substitute	Displays/modifies memory locations. [<cont>] S [W]<addr>[=<expr>][/<expr>]*[,]<cr> [<cont>] S [{I SI LI}] <addr>[=<int number>] [/<int number>]*[,]<cr> [<cont>] S [{SR LR TR}] <addr>[=<real number>] [/<real number>]*[,]<cr> [<cont>] S [T]<addr>[=<BCD number>][/<BCD number>]*[,]<cr>
M Move	Moves the contents of a memory block. M<range> , <dest-addr><cr>
F Find	Searches a memory block for a constant. F<range> , <data><cr>
C Compare	Compares two memory blocks. C<range> , <dest-addr><cr>

MONITOR COMMANDS

Table C-4. Summary of Loader And Monitor Commands (continued)

COMMAND	FUNCTION AND SYNTAX
I Input	Inputs and displays data from input port. [repeat] I [W]<port-addr><cr>
O Output	Outputs data to output port. [repeat] O [W]<port-addr> , <data> <cr>
P Print	Prints values or literals. P [{T S Q}][{<addr> <expr> <literal>}] [, {<addr> <expr> <literal>}]*<cr>
E Exit	Exits the loader program and returns to ISIS-II. E<cr>
* Comment	Rest of line is a comment. * <comment><cr>
B Bootstrap	Bootstraps code from iRMX 86 or 88 file compatible peripherals. B[<pathname>]

INDEX

Underscored entries are primary references.

\$ (default directory) 1-10
:AFD0: and :AFD1: 1-8
:BB: (Byte Bucket) 1-8
:CI: (Console Input device) 1-7
:CO: (Console Output device) 1-7
:LP: (Line Printer) 1-8

abbreviations for command parameters 3-2
Application Loader B-2
ASM86 1-4
ATTACHDEVICE 3-5 to 3-7, 5-3

BACKUP 2-19, 3-8 to 3-14
Basic I/O System B-2
Bootstrap Loader 2-2, B-2
bootstrap loading 1-11, 2-2
BYTE 4-3

comment (command) 2-5
CONNECTION 4-3, 4-5
continuation mark (command) 2-5
CONTROL keys 2-6
COPY command 2-14 to 2-17, 2-18, 3-15 to 3-17
copying system diskette 2-19
CREATEDIR 2-17, 3-18 to 3-19

DATE command 2-8 to 2-10, 3-20
DEBUG 3-21
DELETE 3-22 to 3-23
DETACHDEVICE 3-24
DIR command 2-12 to 2-14, 3-25 to 3-31
directory 1-7
DISKVERIFY 3-32 to 3-36
documentation 6-1 to 6-7
DOWNCOPY 3-37 to 3-39

EDIT 1-4
EPROMs iv, 5-1, 5-9, 5-11, 5-13
exception codes 4-2, A-1 to A-7
exception handlers 1-12
exception-handling system calls 4-6
Extended I/O System B-2

file 1-6
file handling system calls 4-5
file operations from a program 1-8

INDEX (continued)

file operations from terminal 1-8, 2-1
file system 1-5
FORMAT 2-19, 3-40 to 3-44, 5-3
FORTRAN 1-4

hardware 1-2
heirarchical file structure 1-5
Human Interface 3-1, B-2

INCLUDE files (iRMX 86) 1-11
INCLUDE files (UDI) 4-2
inpath-list (command) 2-4
interleave factors 2-19, 3-41, 3-43
invoking commands 2-3 to 2-18
iRMX 86 Operating System 1-1
iRMX 86 PC Operating System 1-1
iRMX 86 PC Release Package iv
iSBC™ 208 Flexible Diskette Controller 1-3, 5-3, 5-6, 5-18
iSBC™ 337 Numeric Processor Extension 5-5
iSBC™ 86/12A Single Board Computer 1-3, 5-7 to 5-9, 5-15
iSBC™ 86/14 Single Board Computer 1-3, 5-10 to 5-11, 5-16
iSBC™ 86/30 Single Board Computer 1-3, 5-12 to 5-13, 5-17
iSBC™ 957B package 1-3

layers of iRMX 86 Operating System 1-1
LIB86 1-4
Library Diskette iv, 1-11
line printer 1-3, 5-4, 5-18
line printer signals (Centronics) 5-4, 5-18
LINK86 1-4
LOC86 1-4
logical name 1-7

manuals 6-1 to 6-7
memory 1-3
memory-management system calls 4-4
monitor 1-12, C-1 to C-12

Nucleus layer B-1
outpath-list (command) 2-4
overlays 4-30 to 4-31
parameters (command) 2-4
parsing 4-20 to 4-21
Pascal1 1-4
pathname 1-7
PL/M-86 1-4
POINTER 4-3
PROG (program directory) 1-10
program (example) 4-44 to 4-48
program loading 1-11
proposition (command) 2-4, 2-5

RENAME command 2-18, 3-45 to 3-47
RESTORE 2-19, 3-48 to 3-54

INDEX (continued)

segment 4-4, 4-9 to 4-10, 4-11
SELECTOR 4-3
SHORT parameter in DIR command 2-14, 3-25
special characters 4-20
STRING 4-3
SUBMIT 3-55 to 3-57
syntax diagrams 3-1 to 3-2
SYSTEM (system directory) 1-10
system call dictionary 4-7 to 4-8
system calls (iRMX 86) B-1 to B-9
system calls (UDI) 4-1 to 4-48
System Diskette iv, 1-9

TIME command 2-8 to 2-10, 3-58
TOKEN 4-3
transparent mode (terminal input) 4-37 to 4-38

UDI 1-5, 4-1 to 4-48
UDI system calls 4-9 to 4-48
Universal Development Interface (UDI) 1-5, 4-1 to 4-48
UPCOPY 3-59 to 3-60

video terminal 1-3
volume 1-6

WORD 4-3
WORK (work directory) 1-10





REQUEST FOR READER'S COMMENTS

Intel Corporation attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

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.



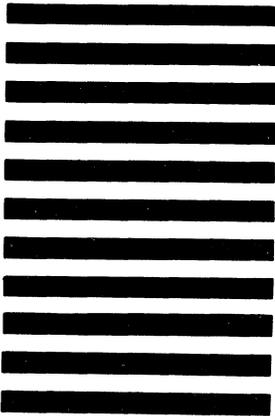
**NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES**

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Pkwy.
Hillsboro, Oregon 97123

O.M.S. Technical Publications





INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 987-8080

Printed in U.S.A.