GETTING STARTED WITH THE RELEASE 5 iRMX™ 86 SYSTEM

Order Number: 145073-001

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

REV.	REVISION HISTORY	PRINT DATE
001	Original Issue	9/82

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 in ASPR 7-104.9(a) (9).

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

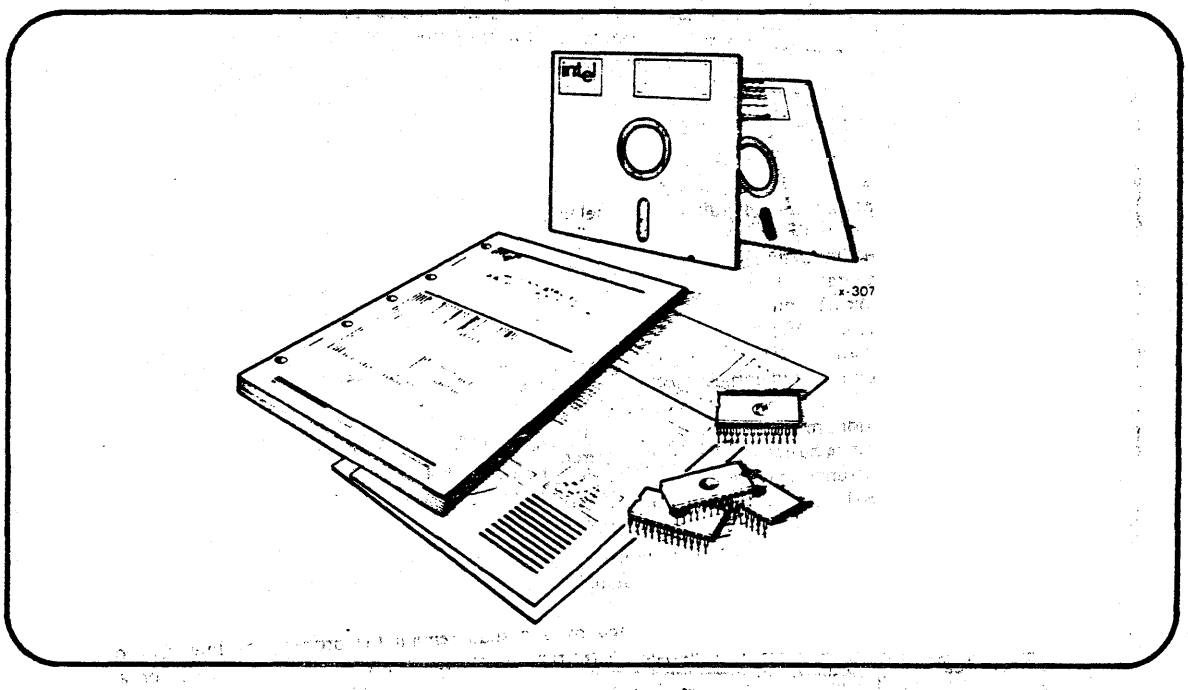
BXP	Insite	iPDS	Micromap
CREDIT	Intel	iRMX	Multibus
i	Intel	iSBC	Multichannel
P-ICE	Intelevision	iSBX	Multimodule
ICE	Intellec	iSXM	Plug-A-Bubble
iCS	inteligent Identifier	Library Manager	PROMPT
iLBX	inteligent Programming	MCS	RMX/80
im	Intellink	Megachassis	RUPI
iMMX	iOSP	Micromainframe	System 2000
			UPI

PREFACE

This manual is a self-contained guide to using the iRMX 86 PC Operating System, (PC = Preconfigured). The Preconfigured Operating System is a ready-to-use version of Intel's configurable, multi-tasking iRMX 86 Operating System. Ready-to-use means that an Intel engineer has already configured the system (put together the subsystems and drivers) so that you don't have to perform the configuration process.

WHAT YOU GET

The Preface Figure shows the contents of the iRMX 86 PC product.



Preface Figure. The iRMX™ 86 PC Product

The iRMX 86 PC Release Package contains:

- The System Diskette, labeled: Preconfigured iRMX 86 Operating System
- The Include File Diskette, labeled: iRMX 86 Include Files
- Four EPROM devices which contain the Monitor and a Bootstrap Loader
- This manual
- One Software Problem Report Form and one Software Registration Card.

GETTING STARTED

To start using the system, you need only:

1. Prepare the hardware on which the system will run. A chapter in this manual explains how to do so.
2. Bootstrap load the system. When you have prepared your hardware, you can bootstrap load the iRMX 86 PC System Diskette.
3. Backup the System Diskette. Once you have bootstrap loaded the system you can make a backup copy of the System Diskette. The System Diskette includes a file of commands (a SUBMIT file) that performs nearly all of the process.
4. Adjust Configuration Files. Although the Operating System is already configured, you can customize the system to match optional hardware that you have in your system. To do so, you change parameters in configuration files that define the characteristics of your particular system. For example, the System Diskette supports only one keyboard terminal, but the iRMX 86 PC System can handle four more terminals using an optional controller board. You can change one value in a file to include other terminals.

If you receive this manual as part of another product, the diskettes and EPROM devices are not necessarily part of the other product.

CONTENTS OF THIS MANUAL

Except for Chapter 5, this manual is written for application programmers who will use the Operating System. Chapter 5 is written for a technician or engineer who assembles the hardware.

Here is how the manual is organized.

Chapter 1	OVERVIEW. This chapter describes the general characteristics of the iRMX 86 PC Operating System. It shows how to initialize (bootstrap load) the system, and how to start using system commands from terminals.
Chapter 2	USING THE SYSTEM. This chapter provides detailed information about the iRMX 86 file system. The chapter contains many examples showing how to use iRMX 86 commands at a keyboard terminal.
Chapter 3	iRMX 86 COMMANDS. This chapter contains individual 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 describes jumpers that must be installed on Intel boards to match iRMX 86 PC characteristics.
Chapter 6	SYSTEM MANAGEMENT. This chapter describes how to set up the system to support multiple terminals and users, and how to prevent users from corrupting the file system.
Chapter 7	DOCUMENTATION. In this chapter, we describe the manuals that relate to the iRMX 86 PC System.
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 describes the sub-systems of the iRMX 86 Operating System and provides a list of "internal" iRMX 86 System Calls. You do not need 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.

RELATED PUBLICATIONS

The following manuals provide additional information that may be helpful. Many of these manuals are described in Chapter 7.

- Introduction to the iRMX™ 86 Operating System, Order Number: 9803124
- iRMX™ 86 Nucleus Reference Manual, Order Number: 9803122
- iRMX™ 86 Basic I/O System Reference Manual, Order Number: 9803123
- iRMX™ 86 Extended I/O System Reference Manual, Order Number: 143308
- iRMX™ 86 Loader Reference Manual, Order Number: 143318

- iRMX™ 86 Human Interface Reference Manual, Order Number: 9803202
- iRMX™ 86 Operators Manual, Order Number: 144523
- iRMX™ 86 Configuration Guide, Order Number: 9803126
- iRMX™ 86 Debugger Reference Manual, Order Number: 143323
- iRMX™ 86 Crash Analyzer Reference Manual, Order Number: 144522
- iRMX™ 86 Programming Techniques, Order Number: 142982
- Guide to Writing Device Drivers for the iRMX™ 86 and iRMX™ 88 I/O Systems, Order Number: 142926
- iRMX™ 86 Terminal Handler Reference Manual, Order Number: 143324
- iRMX™ 86 Disk Verification Utility Reference Manual, Order Number: 144133
- Run-Time Support Manual for iAPX 86,88 Applications, Order Number: 121776
- iRMX 86 Installation Guide, Order Number: 9803125
- ASM86 Macro Assembler Operating Instructions, Order Number: 121628
- Users Guide for the iSBC® 957B iAPX 86, 88 Interface and Execution Package, Order Number: 143979
- iSBC® 86/12A Single Board Computer Hardware Reference Manual, Order Number: 9803074
- iSBC® 86/14 and iSBC® 86/30 Single Board Computer Hardware Reference Manual, Order Number: 144044
- iSBC® 337 Multimodule™ Numeric Data Processor Hardware Reference Manual, Order Number: 142887
- Guide to Configuring Multibus®-Based Systems, Order Number: 144788
- iSBC® 680/681 Multistore User System Package Hardware Reference Manual, Order Number: 162432
- iSBC® 208 Flexible Disk Controller Hardware Reference Manual, Order Number: 143078
- iSBC® 215 Generic Winchester Disk Controller Hardware Reference Manual, Order Number: 144780

- iSBX™ 218 Flexible Disk Controller Hardware Reference Manual, Order Number: 121583
- iSBC® 534 Four Channel Communications Expansion Board Hardware Reference Manual, Order Number: 9800450
- iSBC® 032/048/064 Random Access Memory Boards Hardware Reference Manual, Order Number: 9800488
- iSBC® 016A/032A/064A/028A/056A RAM Boards Hardware Reference Manual, Order Number: 143572

CONTENTS

TABLES

TABLES (continued)

FIGURES

CHAPTER 1. SYSTEM OVERVIEW

Intel has configured the iRMX 86 PC Operating System as an efficient program development base for languages and other software utilities. Figure 1-1 shows the subsystems of the iRMX 86 Operating System. The Preconfigured Operating System includes all of the subsystems shown plus software to support many I/O devices.

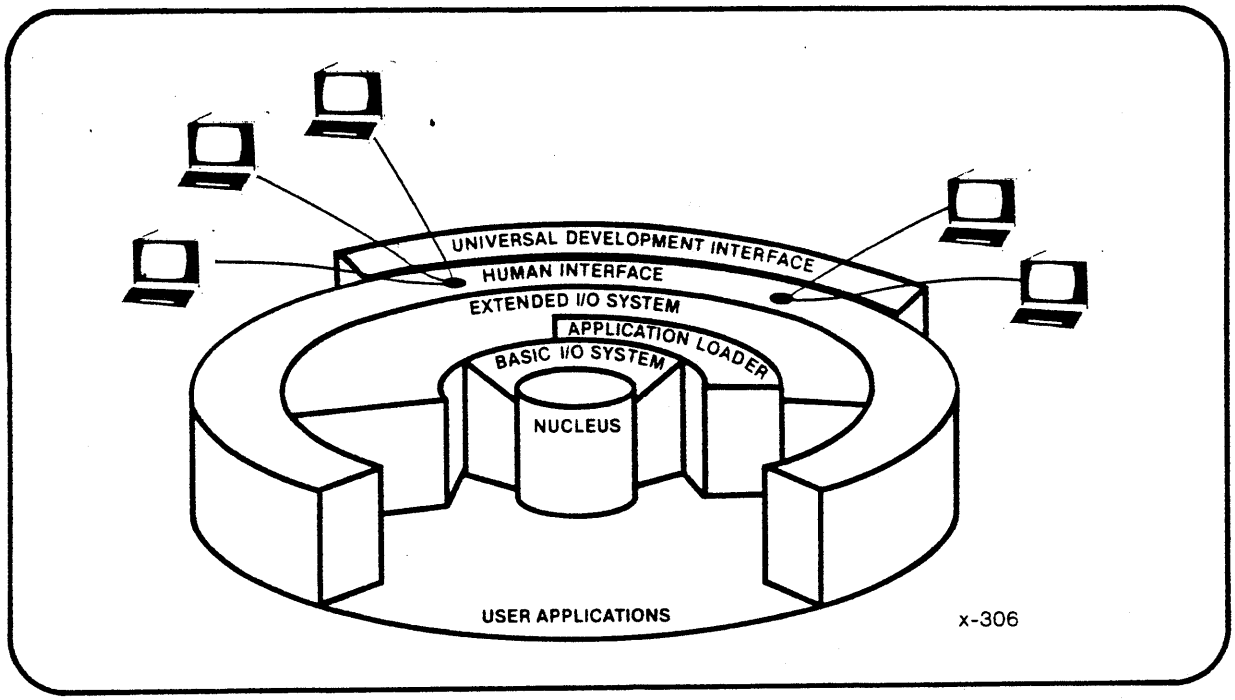


Figure 1-1. iRMX™ 86 Operating System Subsystems

This chapter describes the most important characteristics of the Preconfigured iRMX 86 Operating System. Major subjects of the chapter are:

- Hardware Environment. The iRMX 86 PC System is designed to run on particular hardware; some of the hardware is required and some is optional. The iRMX 86 PC Operating System supports up to five terminals simultaneously, as well as three iSBC 86 Single Board Computers, various disk and diskette drives, and a line printer. The first section of the chapter describes this hardware.

- The UDI and Language Products. A standard software interface -- the Universal Development Interface -- simplifies addition of software packages to your iRMX 86 PC system. Many software packages are available from Intel and from independent software vendors.

- How to Initialize the System. This section shows how to bootstrap load the operating system.

- Starting to Use iRMX 86 Commands. Once the system is initialized, you can enter commands at terminals.

- Setting the Date and Time. This is the first of many examples in the manual using Human Interface Commands.

- The Multi-Access Human Interface. The iRMX 86 PC Operating System can be accessed simultaneously by up to five users at terminals. We describe how the Human Interface manages terminals.

- iRMX 86 Files. This section gives an overview of the Operating System file structure. Chapter 2 describes the file structure in detail.

- Selective Error Processing. You can let the Operating System handle errors or you can let the Operating System detect the error and pass control to your own error-handling routines.

HARDWARE ENVIRONMENT FOR THE iRMX™ 86 PC SYSTEM

In order to configure the iRMX 86 PC system, Intel engineers had to make some assumptions about the Intel boards (single board computers, disk controllers, and so on) that will be used to run the Operating System. This section describes that hardware. Chapter 5, PREPARING YOUR HARDWARE, is a guide to setting up the hardware.

Figure 1-2 on the next page shows a typical iRMX 86 PC hardware system with a flexible disk drive, a Winchester hard disk drive, and multiple terminals. Following the figure is a description of hardware that is required to run the Preconfigured Operating System, and a description of optional hardware.

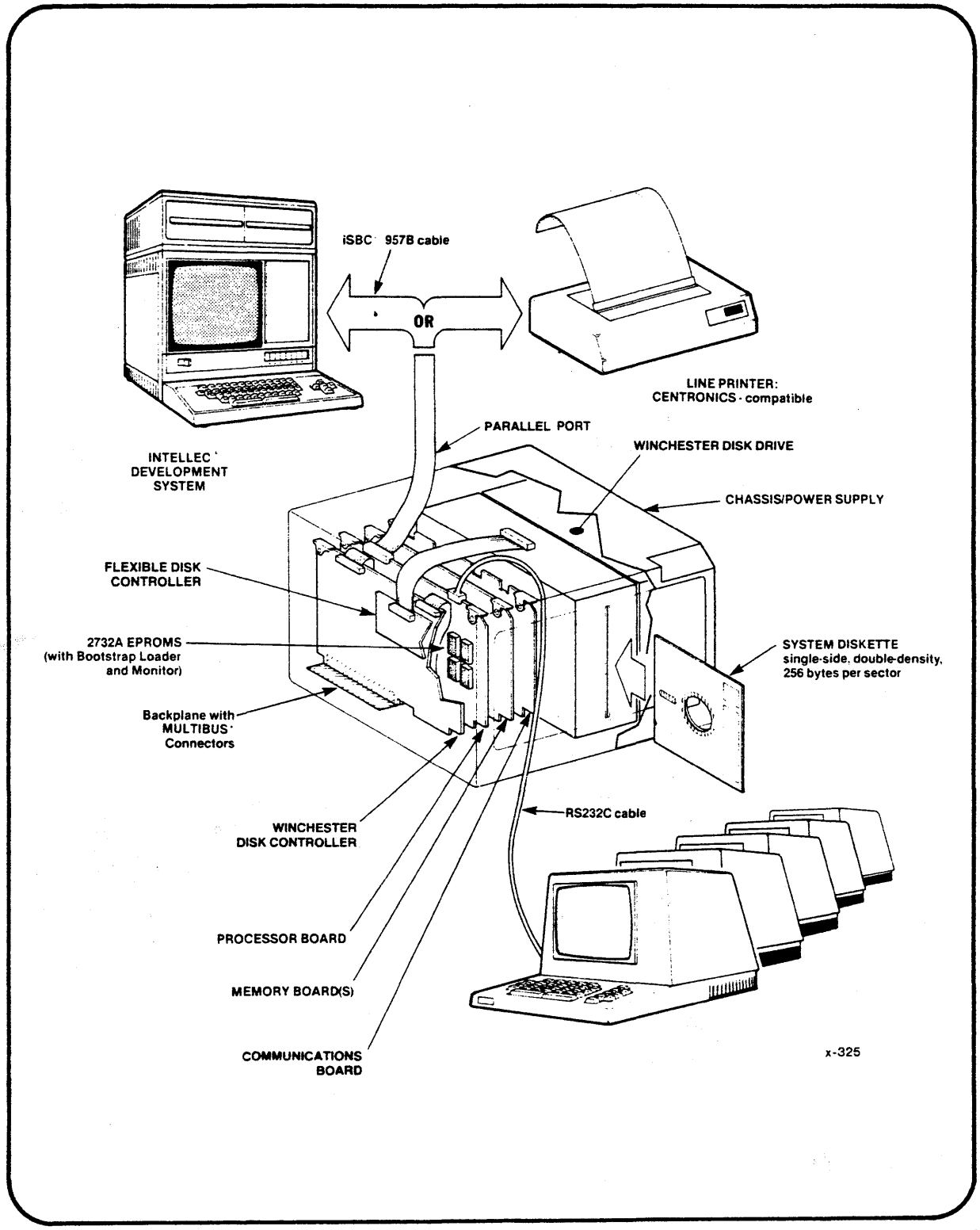


Figure 1-2. iRMX™ 86 PC Hardware Environment

REQUIRED HARDWARE

The Preconfigured iRMX 86 Operating System requires the following hardware:

- Single-Board Computer. An Intel iSBC 86/12A, iSBC 86/14, or iSBC 86/30 Single Board Computer.

- Flexible Disk Drive. A flexible diskette controller with at least one 8-inch drive.

  This disk drive should conform to the size and recording density of the diskettes on which you receive the Preconfigured iRMX 86 Operating System (the format is defined in Chapter 5, PREPARING YOUR HARDWARE).

  Although you can boot and run the system with one flexible diskette drive, you will need at least one other disk drive to do useful work with the system.

- Keyboard Terminal. A keyboard terminal connected to the serial line on your single-board computer.

- Chassis. An appropriate chassis/cardcage/power-supply unit.

- Memory. The Operating System requires 256K bytes of memory.

OPTIONAL HARDWARE

You can include the following optional hardware in your system:

- Four More Terminals. An iSBC 534 Four Channel Communications Expansion Board with one to four keyboard terminals.

- Winchester Disk. A Winchester hard disk drive connected to an iSBC 215 Disk Controller.

- Total of Eight Flexible Disk Drives. Up to four flexible diskettes connected to an iSBC 208 Flexible Disk Controller and up to four flexible disk drives connected to an iSBX 218 Flexible Disk Controller Multimodule. You can use the iSBC 218 Multimodule only if you also have an iSBC 215 Disk Controller.

- Line Printer or Microcomputer Development System. Either a line printer or an iSBC 957B hardware/software package connected through the parallel port on your single board computer. 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.

MEMORY LAYOUT

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

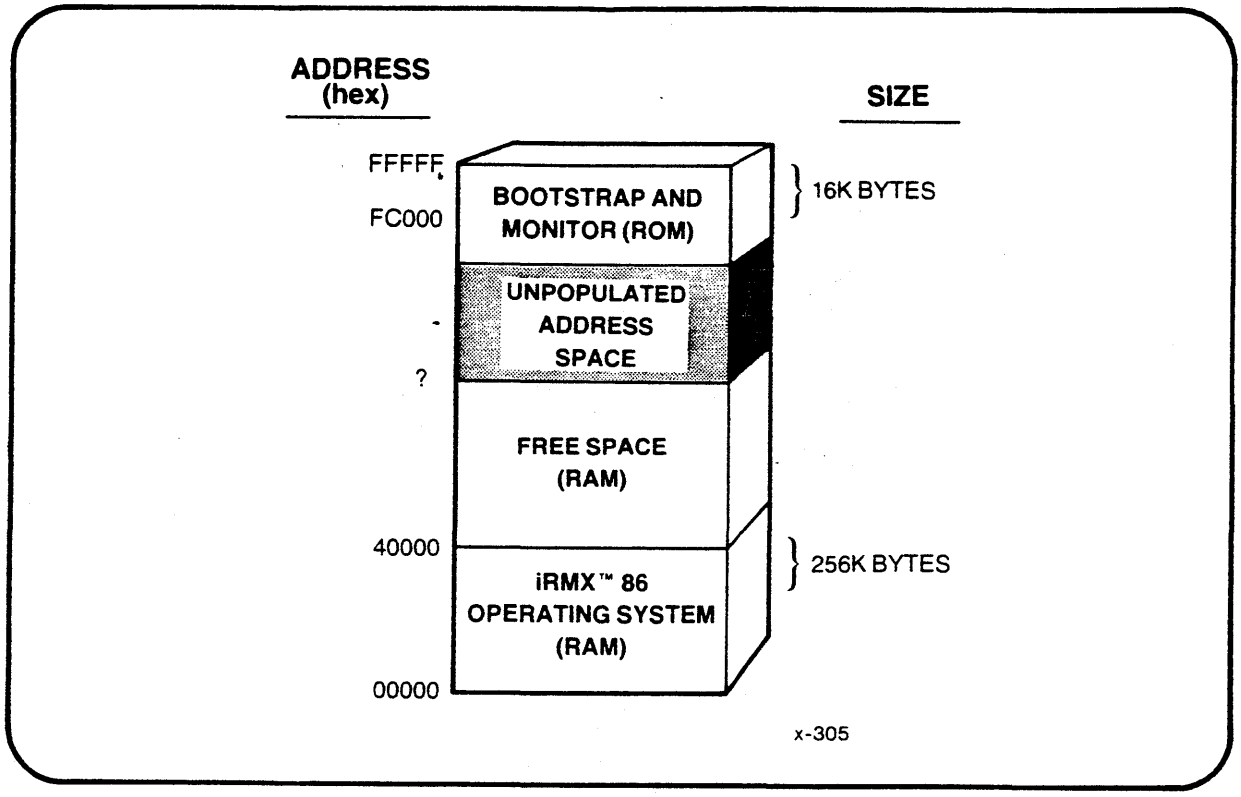


Figure 1-3. Memory Layout of iRMX™ 86 System

THE UDI AND LANGUAGE PRODUCTS

The iRMX 86 PC Operating System includes the Universal Development Interface (UDI), which provides standard system calls for software (your's, Intel's, and vendors') to communicate with the Operating System. Three important advantages to using UDI software are:

- Independence from Operating System Changes. The set of system calls for UDI remains stable regardless of changes in the Operating System.
- Portability. If your software uses only the UDI system calls to communicate with the underlying operating system, you can easily move software packages from one operating system to another.
- Independent Vendor's Software. The UDI standard gives you access to independent software vendors' programs that run on the iRMX 86 Operating System.

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

EDIT	The standard iRMX 86 editor.
ASM86	The 8086/8087/8088 macro assembler.
PLM86	The PL/M-86 compiler.
LINK86	The 8086 Linker, which combines individual 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.
FORTRAN-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.

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 (see Chapter 7 for a list of these manuals.)

HOW TO INITIALIZE THE SYSTEM

This section shows you how to bootstrap load the iRMX 86 PC Operating System. Bootstrap loading (booting) is the process of reading the iRMX 86 PC Operating System into memory from a disk and giving it control of the processor. The bootstrap loader program is in the EPROM components you receive with the iRMX 86 PC product.

The EPROM devices also contain the monitor. With the monitor, you can examine memory, set breakpoints, and (with a hardware package available separately) communicate between your system and an Intellec Microcomputer Development System. Appendix C describes monitor commands.

When the Operating System starts running, it initializes itself. Then it is ready to accept commands typed from a keyboard terminal. The next section describes how to use a terminal (start an interactive job).

If you have installed the EPROM components on your single board computer, bootstrap the system as follows (refer to Figure 1-4.)

1. Turn on power to the disk drive, processor and to the system terminal -- the one connected to the single board computer.

2. Insert a copy of the System Diskette into Disk Drive 0. We show you later how to copy the Diskette you receive.

3. When you see a series of asterisks appear on the display screen, type a single uppercase

   U

   The system repeatedly sends characters to the screen until you type a U. The U is not echoed to the screen. What character is sent to the screen depends upon the baud rate, and we show asterisks (*) in our example -- the character displayed at 9600 baud. A hand points to where the U is typed.

4. You will see a message identifying the Monitor, followed on the next line by a prompt of "." (period).

5. Respond by typing a single

   B

   meaning boot. You can also use a lowercase b.

6. Now the Bootstrap Loader reads the Operating System into memory from your diskette, and passes control to it. (This usually takes less than a minute.)

Figure 1-4 shows the screen display when the system is booted.

If your system has a button connected to the RESET line on the single board computer, you can re-boot by using the button. The system will again display asterisks, and you continue from step 3.

CAUTION

To prevent destroying data on your disks when re-booting, allow 2 seconds to elapse after the last I/O operation from commands or programs that access the disk. This also applies when removing diskettes from drives and when turning off power.

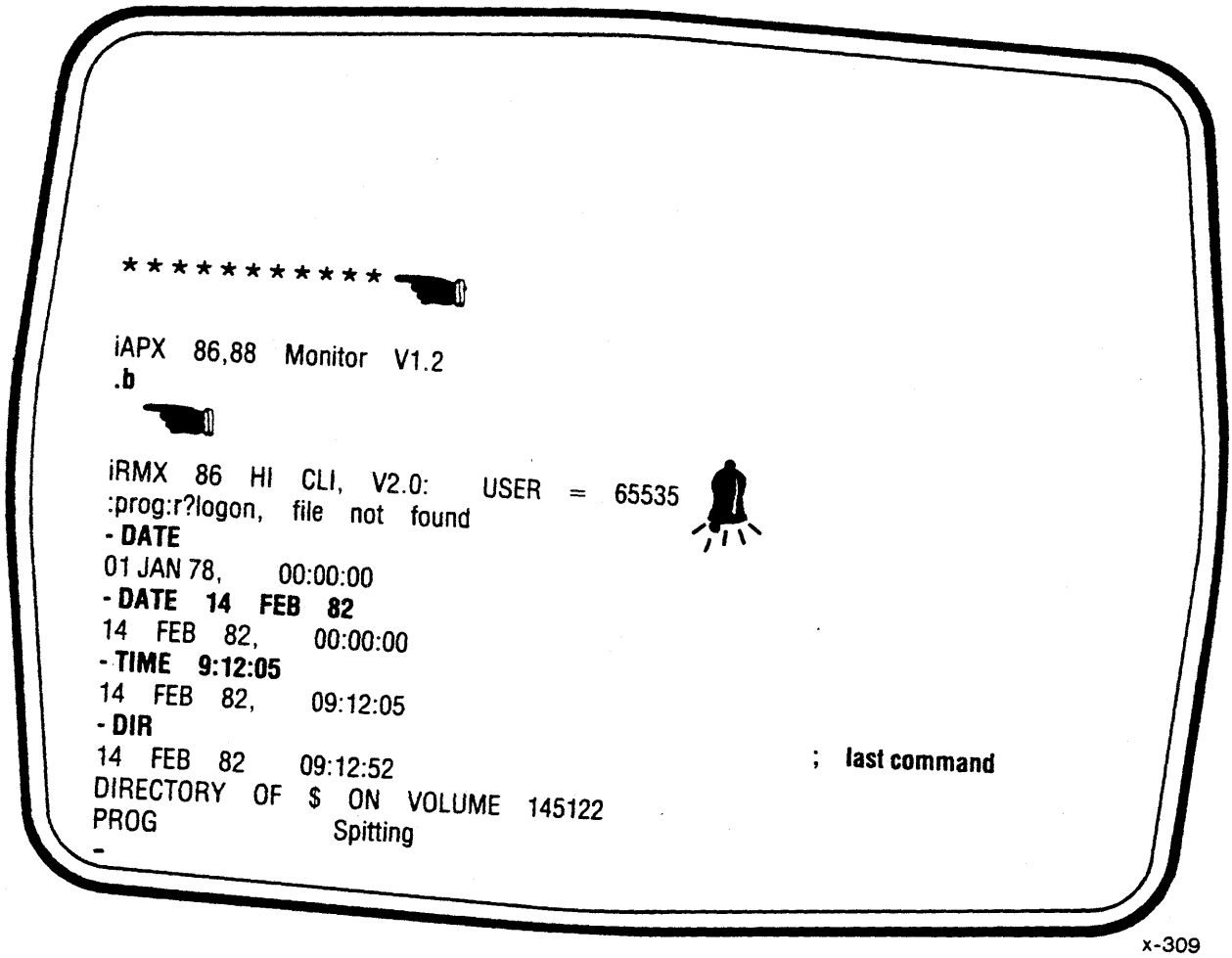


Figure 1-4. Initializing the System

NOTE

In illustrations of video screens, what the operator types (console input) is shown in this typeface.

Messages displayed by the system (console output) are shown in this typeface.

Input and output lines in illustrations are not proportional to an actual video display.

STARTING TO USE A TERMINAL

When you have successfully booted the system, you can begin entering commands at any terminal. That is, you can start an interactive session. How to do so is shown in Figure 1-4 and explained following the figure. From this point, the system terminal is used the same as any other terminal.

Here is how to start.

1. Type an uppercase U; it is not seen on the screen. A hand on Figure 1-4 shows where it is typed. The system reads the "U" character to determine the data baud rate of your terminal, and then adjusts the computer rate to match it -- a process called automatic baud-rate selection.

2. The iRMX 86 Command Line Interpreter (CLI) begins running and displays its header (and the terminal bell rings):

   ```
   iRMX 86 HI CLI, V2.0: USER   65335
   ```

   In Figure 1-4, V2.0 is the version number of the CLI, and 65535 is the user ID assigned to the terminal. While you are using a terminal, a user ID is your identification to the Operating System. The next section will deal with user IDs and why they are important.

3. Next, the CLI searches for the logon file. Each user ID can have a unique logon file. If you do not have a file named R?LOGON, the system will display a message such as:

   ```
   :prog:r?logon, file not found
   ```

   The logon file is optional. If it finds a logon file for a terminal, the CLI automatically SUBMITs the file (reads each line in the file and executes the line as a command). Using a logon file, you can start processes automatically when the Operating System first recognizes your terminal. The system recognizes the name :prog:r?logon as a logon file. Chapter 2 explains iRMX 86 file names.

4. After processing the commands in the logon file, the CLI issues its prompt (-) and returns control to you. At this point you can enter Human Interface commands and invoke programs.

SETTING THE DATE AND TIME

In this section, we show how you can set the system time and date. The TIME and DATE commands are shown first because it is a good idea to set the system time and date as soon as the system is bootstrap loaded. In Figure 1-4, the operator typed DATE in response to the first Human Interface prompt (-).

Without parameters, the DATE command simply displays the date. In this case, the system displays the default date and time that indicates it hasn't been set since the system was booted.

The next line shows the operator setting the date to Valentine's Day of 1982, and the system responds by displaying the new date and old time.

The next lines shows a similar sequence for the TIME command. System time is set to 12 minutes and 5 seconds after 9 AM, and the system confirms the new time.

Finally, the command DIR is typed, and the system shows the files that are in the user's directory. The next chapter describes the DIR (directory) command.

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 by commands or with system calls, the time remains at 00:00:00.

2. When you display the contents of a directory, the line with the date and time will not be shown. Note that the time and date is displayed in the example DIR command, because the time was set.

If you type a CTRL/z in response to the Human Interface prompt (-), you will reinitialize the interactive job and take up at step 2. (CTRL/z is the letter z typed while holding down the CONTROL key.) If you turn your terminal off but the system remains running, you can turn the terminal back on and continue as though you had not turned it off. Or you can turn it back on and type CTRL/z. Re-initializing the interactive job destroys any logical names you have created during the session. (Logical names are defined in the next chapter.)

THE MULTI-ACCESS HUMAN INTERFACE

The iRMX 86 Human Interface (Figure 1-1) allows several users to access the Operating System at the same time. Each terminal has an associated interactive job and user ID, as explained next.

INTERACTIVE JOB

When the system is initialized, the iRMX 86 Human Interface assigns an interactive job to each terminal. The interactive job consists of an identifier (user ID), a program that runs immediately (initial program), and an area of memory in which the programs can run. User IDs and the initial job are described in the next two sections.

The Operating System also provides a names for the individual terminal keyboard and terminal screen being used. These are logical names and are described in the LOGICAL NAMES section of Chapter 2.

USER ID AND OWNER ID

As part of creating the interactive job, the Human Interface assigns a user ID to the terminal. This user ID is your "identity" in the system. It determines your access to files and to devices. When you create files, you are the owner of the file and control the file.

Two special user IDs are:

- WORLD — The user ID called WORLD has a numerical value of 65535 (0FFFFh). WORLD represents all users of the system. Every user has complete access to files owned by WORLD; files owned by WORLD can be read, written, and deleted by any other user. So if you wish to restrict access to a file, your user ID should not be WORLD.

- System Manager — User ID 0 designates the system manager. The system manager can read all files on the system and can list every directory on the system. The system manager can also change the access of any file on the system. The iRMX 86 PC System is delivered with files that are owned by the system manager.

  Chapter 6 of this manual is about the system manager and explains such things as how to add new terminals to the system, how to change the user ID of a terminal, and how to copy the System Diskette.

iRMX™ 86 FILES

A basic function of the iRMX 86 PC Operating System is to provide a file system. You can manipulate these files with iRMX 86 commands invoked from a terminal, as described in Chapter 2. Programs access files with the UDI system calls described in Chapter 4.

To manipulate iRMX 86 files and directories at a terminal, you run programs as commands. The iRMX 86 Operating System provides commands to perform operations that are usually necessary in a development system. A few of these are:

- COPY, which copies files.
- DIR, which displays the contents of a particular directory.
- CREATEDIR, which creates a new file directory.
- SUBMIT, which automatically executes other commands contained in a file.
- FORMAT, which prepares a new disk, diskette, or other mass storage volume for file data.
- SUPER, which assigns a special user ID to the terminal, allowing unlimited access to files.

Programs must be able to manipulate files. An assembler, for example, must open and read source files, and it must create object files. Programs can read, write, delete, and otherwise deal with files by using UDI system calls. Chapter 4 tell how you can use UDI system calls that are provided with the iRMX 86 PC Operating System.

SELECTIVE ERROR PROCESSING

When a program or user at a terminal causes an error that the Operating System detects (for example, a program might request memory that is not available) the iRMX 86 PC Operating System's default exception handler terminates the program and displays 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 multi-tasking, multi-user operating system used for many types of systems. This chapter has discussed features that directly relate to using the iRMX 86 PC Operating System.

The next chapter explains how to use Human Interface commands at a terminal and how to run your programs as commands. Much of Chapter 2 is a description of the iRMX 86 file structure, because most Human Interface commands (the subject of Chapter 3) and most UDI system calls (Chapter 4) are used for file operations.

***

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.

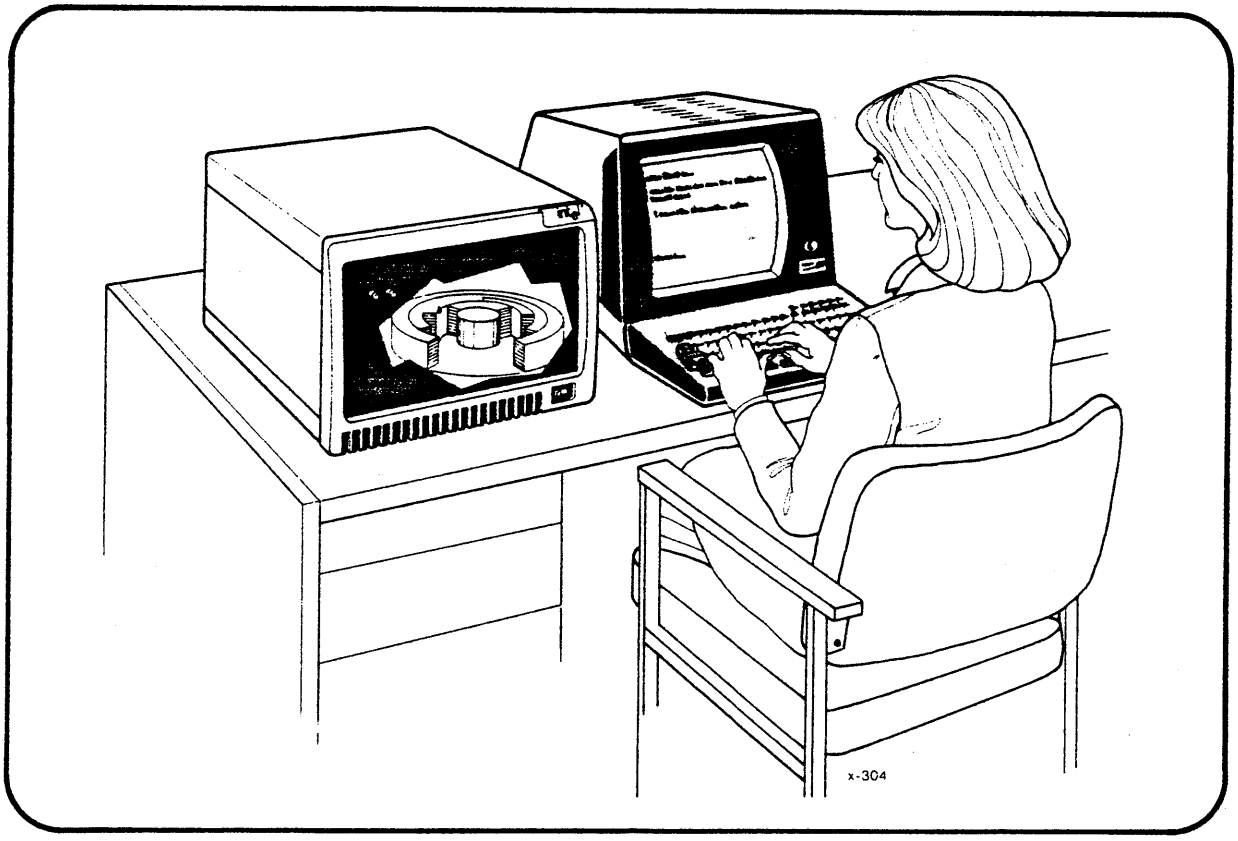


Figure 2-1. Using The iRMX™ 86 Operating System From A Terminal

In this chapter, we describe how to use the Operating System by using many examples of iRMX 86 commands and system responses. Chapter 3 describes in detail all of the commands that Intel provides with the iRMX 86 PC System.

This chapter is organized as follows:

- INVOKING iRMX 86 COMMANDS. This is general information about how to use iRMX 86 commands.
- SYNTAX OF COMMANDS. This describes, with examples, the general syntax of iRMX 86 commands. It also shows an example of the type of syntax diagram that is used in Chapter 3 to describe commands.
- USING iRMX 86 FILES. This section of the chapter introduces the important concepts about the iRMX 86 file system, and shows examples of commands that manipulate files.
- MORE ABOUT iRMX 86 COMMANDS . This explains capabilities of the iRMX 86 PC System that you will want to use after you are familiar with the basic concepts.

INVOKING iRMX™ 86 COMMANDS

This section describes how to invoke Human Interface commands.

EXAMPLE: DIR AND COPY COMMANDS

Figure 2-2 shows examples of the DIR (directory) and COPY commands. The individual commands are described on the next page.

```
-DIR                                    ; first command
14 FEB 82    09:12:52
DIRECTORY OF    $    ON VOLUME    143774
PROG            Spitting

-COPY Spitting TO Image                 ; second command
Spitting copied TO Image
-DIR                                    ; third command
14 FEB 82    09:13:12
DIRECTORY OF    $    ON VOLUME    143774
PROG            Spitting          Image

-DIR SHORT ONE                          ; fourth command
SHORT, file does not exist
-DIR :$: SHORT ONE                      ; fifth command
14 FEB 82    09:13:27
DIRECTORY OF    $    ON VOLUME    143774
NAME            AT              ACC          BLKS          LENGTH
PROG            DR              DLAC            1            240
Spitting                        DRAU            1             54
Image                           DRAU            1             54
-               3 FILES                     3 BLKS     348 BYTES
-
```

x-310

Figure 2-2. Example Human Interface Commands

NOTE

To help explain these example screens, we show comments ("; first command," etc.) as part of commands the operator types. A command line can have a comment at the end of the line preceded by a semicolon. Comments are typically used with SUBMIT files (described later). You probably won't type comments while entering commands interactively, but doing so is OK.

first command — DIR shows all of the files in a directory named :$:. This is your directory, your default directory. We will explain more about this and other directories later. The system responds by showing that :$: contains two files or directories (we can't tell which) called PROG and Spitting.

second command — This COPY command creates a new copy of the file Spitting, and calls it Image.

third command — A DIR command shows that :$: now contains three files.

fourth command — Without any listing options, the DIR command lists only file and directory names, as in the first command in Figure 2-2. We don't know whether each entry in the directory is a file or another directory (iRMX 86 directories can themselves contain other directories). So we use a parameter, SHORT, to ask for a directory display that shows more information about the contents of the directory. We also ask for a ONE-column listing format. (Chapter 3 describes every DIR listing format.)

The fourth command, as typed, makes the system think that we want to list a directory named SHORT. This shows what happens when you type a command that the Human Interface cannot understand. The system responds by displaying an error message.

fifth command — To correct this, we identify the directory :$: before the parameters SHORT and ONE. The system responds by again showing the files in the default directory (:$:), but also shows that PROG is a directory (DR in the ATTributes column) and that Spitting and Image are files (blank in ATT column). The size of each is also shown as blocks and as bytes.

The ACCess column describes the access allowed for the user ID at this terminal.

For directories	For files
D = Delete	D = Delete
L = List	R = Read
A = Append	A = Append
C = Change	U = Update

The owner (user ID that created the file or directory) has full access to the file. You can selectively allow access to files that you own, using the PERMIT command described in Chapter 3.

SYNTAX OF COMMANDS

This section describes the general structure of a command. A command can contain the following elements:

command-name inpath-list preposition outpath-list parameters

Except for the command name, the elements in a command are optional for some commands and required for others, or simply don't apply to certain commands. For example, the DATE command does not have an inpath-list or outpath-list, but does have an optional parameter (QUERY).

A RETURN key or LINE FEED key terminates each command.

The meaning of each element is described here.

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 disk and executes the command. In Figure 2-2, the first command name is DIR.

inpath-list — One or more pathnames (iRMX 86 file names are called pathnames) 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 second command in Figure 2-2, the inpath-list is

Spitting

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 second command in Figure 2-2, 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 second command of Figure 2-2, the outpath-list is:

Image

parameters — Most commands have a default form but also offer one or more optional ways in which 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 fifth command in Figure 2-2, the parameters are SHORT and ONE.

The Human Interface makes no distinction between cases when it reads command line items, so you can enter elements of a command line in uppercase characters, lowercase characters, or a mixture of both.

Figure 2-3, a "railroad track" syntax diagram for the COPY command, gives you a preview of how Chapter 3 describes commands. Each command in Chapter 3 includes one of these diagrams. The diagram is read left to right.

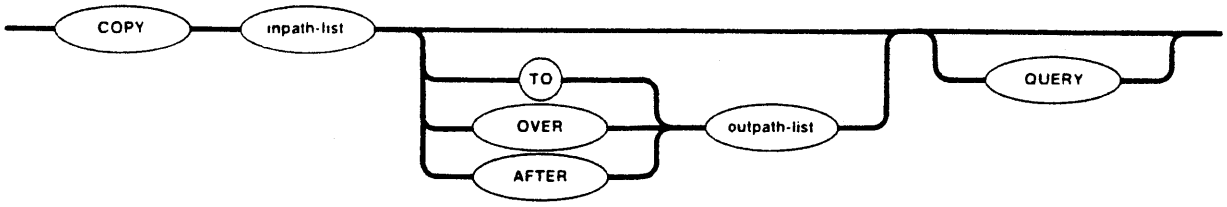


Figure 2-3. Syntax Diagram of COPY Command

USING iRMX™ 86 FILES

One important use of Human Interface commands is to manipulate files. To use Human Interface commands and to use the UDI system calls, you need to understand the iRMX 86 file structure.

If you are reading simply to understand the major characteristics of the iRMX 86 PC System, you may wish to skim through the following discussion of the file system. Later, when you are ready to use the system, you can read the details in this section. Briefly, the section describes:

TYPES OF FILES — There are three: physical files, named files, and stream files. Named files, the most frequently used file type, provide file trees on a device.

FILE TREES — The iRMX 86 Operating System supports an open-ended structure of files and directories starting with a root directory. Directories can contain both files and other directories. Figure 2-4 in the following section shows a file tree.

PATHNAMES — These are names for files in a named file structure: a pathname describes a path from one directory through each lower-level directory to the file. For example, in Figure 2-4, the pathname of the file with the name r?logon is:

```
:SD:USER/1/PROG/r?logon.
```

LOGICAL NAMES — Logical names simplify the way you refer to a path through the file tree, and the way you refer to devices. The Operating System has some built-in names for your convenience.

TYPES OF FILES

Most of this discussion of iRMX 86 files is about named files, because that is the type used most frequently with Human Interface commands and with the UDI system calls.

The named file structure divides data on mass storage devices into individually-accessible files and directories, as shown in Figure 2-4. Commands and programs refer to these files by name when they want to access information stored in them.

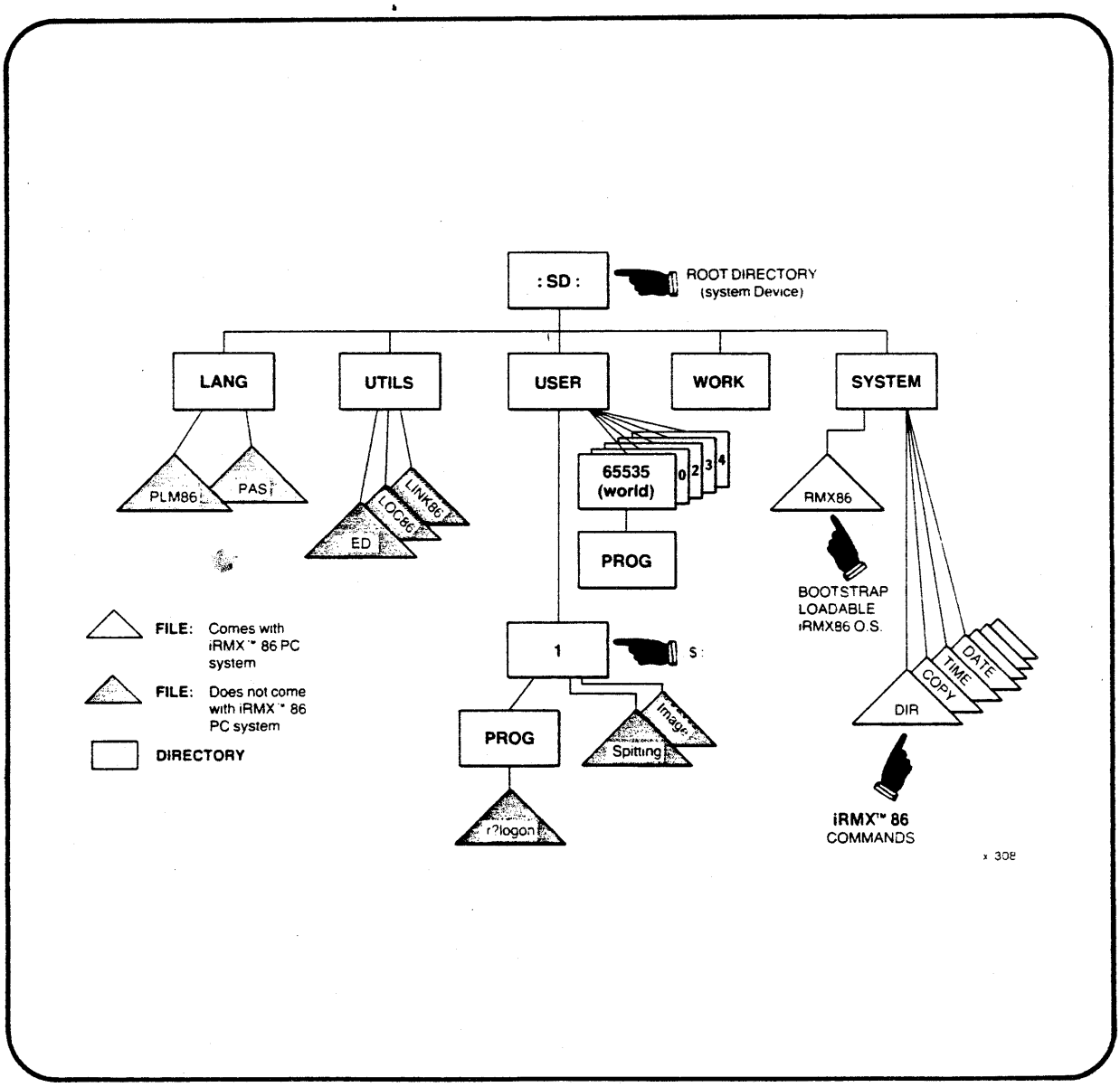


Figure 2-4. An iRMX™ 86 Named File Structure

The other types of files are:

Physical Files	A physical file is an entire I/O device, represented as a single file. The Operating System accesses backup volumes (like tapes), as well as line printers and terminals as physical files.
Stream Files	Stream files are software mechanisms that can be used only with iRMX 86 Basic I/O or Extended I/O system calls.

FILE TREE STRUCTURE

An iRMX 86 file tree has two kinds of files: data files and directories. Data files (shown as triangles in Figure 2-4) contain programs and the information that you manipulate with programs (for example, inventory control programs, text, source code and object code). Directories, (shown as rectangles in Figure 2-4) contain pointers to named files or other directories.

The iRMX 86 Operating System allows you to have multiple directories on a mass storage device. Instead of having a single directory containing all of the files on the device, you can group files into several directories. Duplicate file names are permitted unless the files reside in the same directory. For example, the file tree in Figure 2-4 contains two directories named PROG. However, they are unique because each resides in a different directory.

The uppermost point of each file tree is a special directory called the root directory, which contains all other files and directories on the device. A file tree cannot extend to more than one volume, and one device must be the system device (root directory named :SD:).

PATHNAMES

In a file tree, each file and directory has a unique path connecting it to the root directory. When you want to perform a file operation, you must specify not only the file's name, but the path through the file tree to the file. This is called the file's pathname. For example, in Figure 2-4, :SD:USER/1/PROG/r?logon is the pathname of the logon file for user ID 1. The path from the root directory (:SD:) to the file goes through directory USER, through directory 1, through directory PROG, and finally stops at data file r?logon.

Slashes (/) separating individual components of the pathname tell the Operating System that the next component is down one level. You can use a circumflex (^) between path components to refer to the next higher level.

A later section, LOGICAL NAMES, shows how you can assign a short-hand name to a file for convenience.

EXAMPLE: DIRECTORY OF iRMX™ 86 PC FILES

Figure 2-5 shows some commands that operate on the iRMX 86 PC file structure shown in Figure 2-4. Figure 2-4 shows a recommended file structure for the iRMX 86 PC system. This is the structure of the System Diskette that Intel delivers to you as part of the iRMX 86 PC product. Some files and directories shown in the figure are not provided with the iRMX 86 PC system, but are included for discussion purposes.

The example commands shown in the remainder of this chapter represent a single terminal session. That is, each screen takes up where the previous one left off. Figure 2-5 and the following screens in this chapter assume that the terminal has been assigned user ID 1. Some example screens introduce concepts or terms that are only briefly explained with the example, but are described in detail later.

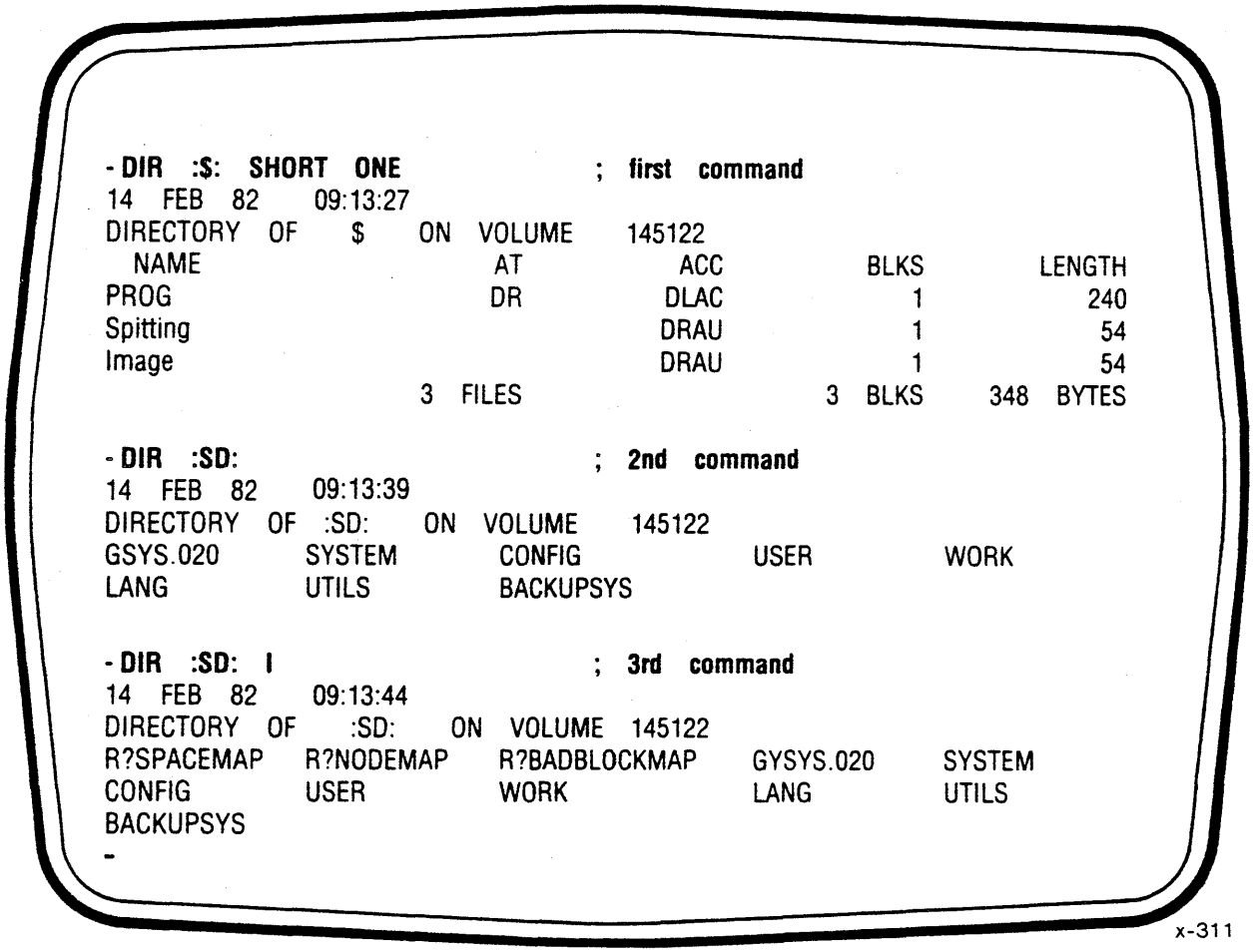


Figure 2-5. Directory Listings of iRMX™ 86 PC Files

The effect of each command is described here (commands are identified by the comment at the end of the command line).

first command

This is the same command seen at the end of the previous example screen, Figure 2-2.

second command

This command shows how to list the directories and files in the root directory of the system device. This DIRectory listing reveals three files not shown in Figure 2-4:

- GSYS.020, a file that identifies the iRMX 86 PC Operating System. A later example shows the contents of this file.
- BACKUPSYS, which is used to backup the system diskette. How to backup the system diskette is explained in Chapter 6.
- CONFIG, a directory that is used for system management. Chapter 6 describes the files in this directory.

The root directory (:SD:) also contains some "invisible" files: files that the Human Interface does not normally show in directory listings. To the Operating System, any file name starting with R? is invisible. For example, the logon file for user ID 1 -- r?logon in Figure 2-4 -- is invisible.

third command

This is the same command, but with an added parameter, I (meaning Invisible). The command could have been typed without abbreviating Invisible:

```
DIR :SD: INVISIBLE
```

Command parameters may be abbreviated. You abbreviate a command parameter by typing enough of the parameter name to make it unique among other parameters for the command. No other DIR parameter starts with the letter I, so I is a valid abbreviation for "Invisible". Remember that you can use either uppercase or lowercase letters for elements in a command.

The three invisible files in the root directory are: R?SPACEMAP, R?NODEMAP, and R?BADBLOCKMAP.

These three files were created when the disk was formatted, and are used by the iRMX 86 Operating System. A later example shows how to format a disk or other mass storage volume.

EXAMPLE: SUPER COMMAND AND CHANGING THE DEFAULT DIRECTORY

```
-ATTACHFILE :SD: AS $                                   ; first command
:SD: , attached as :$:
-DIR                                                    ; 2nd command
14 FEB 82    09:14:26
DIRECTORY OF   $   ON   VOLUME   145122
GSYS.020          SYSTEM          CONFIG          USER          WORK
LANG              UTILS           BACKUPSYS

-COPY GSYS.020 TO :CO:                                  ; 3rd command
:SD:GSYS.020, READ access required
-SUPER                                                  ; 4th command
enter password:
super- COPY GSYS.020 TO :CO:                            ; 5th command
Preconfigured iRMX 86 Operating System V2.0 (143774)
GSYS.020 copied TO :CO:
super- EXIT                                             ; 6th command
-
```

x-312

Figure 2-6. Examples: Changing Default Directory and SUPER Command

first command — This command changes the default directory for user ID 1, so that the default directory is now the root directory (:SD:). You can change your default directory whenever it is convenient to do so. One reason to change your default directory is to do work (create or edit files, run programs, etc.) in a different directory. If you don't want to type the pathname of the directory each time you refer to it, you can make the directory your new default directory.

Immediately following the explanation of this screen is a discussion of the important facts about default directories and the name :$:.

2nd command — This DIRectory shows that the default directory has indeed become :SD:. The effect of the command is the same as the command: DIR :SD: shown in Figure 2-5.

3rd command — This command attempts to display the contents of the system identification file (GSYS.020) on the terminal screen. The system responds with an error message explaining that this user ID doesn't have READ access to this file. The owner of a file can selectively restrict or allow access to the file using the PERMIT command, a feature called access control.

4th command — To assume the user ID 0 (user ID 0 has full access to all files on the system), the operator uses the SUPER command and the system prompts for a password. When the password is typed, it is not displayed on the video screen. The hand in the illustration shows where the password is typed.

Although the Human Interface doesn't normally distinguish between uppercase and lowercase letters, the password is an exception; it must be typed exactly as defined. The system manager is responsible for defining the password.

When the password is successfully typed, the system prompt changes to super- and the user ID becomes zero.

5th command — This displays the contents of GSYS.020 on the screen. The command could have been typed

```
COPY GSYS.020
```

because if you do not type a preposition and output pathname with the COPY command, it assumes TO :CO: (to your console output device).

The file GSYS.020 is an identification file, and contains one line of text. This file should be be kept on copies of the system disk, so that if you need to contact Intel regarding the Operating System, you can positively identify the product.

6th command — The EXIT command returns the user ID to its original value, and the normal prompt (-) is displayed on the next line.

DEFAULT DIRECTORY

When the system is initialized (bootstrap loaded), the Human Interface assigns a user ID to each terminal, and each user ID is assigned a default directory. The name of the default directory is :$:. This means that when user ID 65535 refers to :$:, the Human Interface uses :SD:USER:65335. When user ID 1 refers to :$:, the Human Interface uses :SD:USER/1.

The iRMX 86 Operating System provides default directories for your convenience. When performing file operations (using commands, using interpreters, compilers, or your own programs) you can specify the name only of files that are in your default directory, rather than the complete pathname of the files.

Being able to change your default directory (see the first command in Figure 2-6) extends the value of this feature.

NOTE

This manual uses the term :$: to refer to the default directory for a particular user ID. This is to remind you that it is a logical name (logical names are described in the next section). But for speed or convenience you can use $ without colons. For instance:

```
DIR $ S
```

gives a SHORT directory listing of the default directory.

A change in your default directory assignment remains in effect until the system is re-initialized or until you change the assignment by using another ATTACHFILE command. You can easily return :$: to its initial assignment by typing the command ATTACHFILE without parameters. This is shown in a later example.

Intel delivers the iRMX 86 PC System with unique user IDs assigned to each terminal. But you may change these assignments, and may also specify which directory is initially assigned as a user ID's default directory. How to do so is explained in Chapter 6, SYSTEM MANAGEMENT.

LOGICAL NAMES

Although you can always identify files with pathnames, you can also create logical names for files and directories, and for devices. This makes it easy to refer to files and devices that you use frequently. For example, you could assign the name :SOURCE: to the file :SD:USER/PROG/PASC/MYPROG.PAS. You may then refer to the file as :SOURCE: (you can still refer to it by its pathname).

You establish logical names for devices with the ATTACHDEVICE command, and for files and directories with the ATTACHFILE command. The Operating System creates certain logical names when it is initialized. (:SD: and :$: are two of these names; the others are described later.) You can have both ATTACHDEVICE and ATTACHFILE commands in a logon file if you want to automatically name devices and files when the system is booted. A later example shows how to create a logon file.

A logical name contains 1 to 12 ASCII characters surrounded by colons (the colons do not count in the 12); the value of each character must be between 020h and 07Fh inclusive (the printing characters and the space). Logical names cannot include the colon (:), slash (/), up-arrow or circumflex (^), asterisk (*), or question mark (?).

Logical Names for Devices

The Preconfigured iRMX 86 Operating System creates certain logical names for devices when the system starts running. You can establish other logical names for new or existing devices by invoking the ATTACHDEVICE command. By using a device logical name as the first element in a pathname, you can refer to any file on any device. Suppose your system contains two flexible disk drives for which you have established logical names :F0: and :F1: (in other words, you used the ATTACHDEVICE command to attach the devices as :F0: and :F1:). If you have a diskette containing the file DEPT2/HARRY, you could place the diskette in drive :F0: and access the file with the pathname:

```
:F0:DEPT2/HARRY
```

If you put the same diskette in drive :F1:, you could access the file by specifying the pathname:

```
:F1:DEPT2/HARRY
```

For devices containing named files, the device logical name is also a name for the root directory on that device. If you enter the command

```
DIR :F1:
```

you will see a listing of the files in the root directory of :F1:.

These logical names for devices are already defined when the iRMX 86 PC system is initialized:

:SD:	The system device. This logical name refers to the disk drive from which the Bootstrap Loader read the Operating System file. :SD: refers to the same device for all users on the system.
:CI:	The terminal keyboard (or command input). Each user's :CI: refers to the terminal associated with that user.
:CO:	The terminal screen (or command output). Each user's :CO: refers to the terminal associated with that user.
:LP:	The logical name of the line printer.

Also, the Operating System provides two useful simulated devices (they do not exist as actual I/O devices) identified by these logical names:

:BB:	An infinite sink (byte bucket). Anything written to :BB: disappears, and a read from :BB: returns an end-of-file.

A later example demonstrates how the ATTACHDEVICE command makes a device known to the system by a logical name.

Logical Names for Files

A logical name for a file (or named file directory) provides a shorthand way of accessing that file. For example, suppose you have a file that resides several levels down in a file tree, such as:

```
:F1:DEPT1/TOM/TEST-DATA/BATCH-2
```

where :F1: is logical name for the device that contains the file. You might find it inconvenient to continually enter so many characters. If so, you can establish a logical name for this pathname, such as :BATCH:. (This is the same as saying that you attached the file with the logical name :BATCH:.) Then, whenever you want to refer to the file in a command, you can specify the logical name instead of the pathname.

If a logical name refers to directories instead of data files, you can use the logical name in the prefix portion of a pathname. For example, consider the same pathname:

```
:F1:DEPT1/TOM/TEST-DATA/BATCH-2
```

Suppose you have attached the pathname :F1:DEPT1/TOM/TEST-DATA as logical name :TEST:; therefore it is a logical name for the directory TEST-DATA. To refer to file BATCH-2, you could enter:

```
:TEST:BATCH-2
```

Logical names for files come into existence in two ways. One way is for you to invoke the ATTACHFILE command. The other way is for the Operating System to create them. The iRMX 86 PC Operating System establishes certain logical names for files and directories, and these are described next.

SYSTEM DEVICE (:SD:). Not only is this the logical name for the system device, as described in the previous section, but :SD: is also the logical name for the root directory that contains all other directories and files on the system device.

SYSTEM DIRECTORY (:SYSTEM:). This directory contains the following directories and files:

- iRMX 86 COMMAND PROGRAMS. When you invoke 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. All of these commands are described in Chapter 3.

- OPERATING SYSTEM. The file :SD:RMX86 contains the iRMX86 PC Operating System; this is the file that is read in by the Bootstrap Loader.

DEFAULT PREFIX (:$:) AND HOME DIRECTORY (:HOME:). An earlier section describes the default directory logical name: :$:. In summary, :$: can refer to a different directory for each user ID. If you don't specify a logical name at the beginning of a pathname, the Operating System assumes that the file is in the directory corresponding to :$:.

When you first start using the Human Interface, the logical names :HOME: and :$: represent the same directory. You can easily re-establish your original :$: logical name using the command ATTACHFILE and no parameters.

PROGRAM DIRECTORY (:PROG:). You can use this directory for programs that you write. The Operating System automatically finds and runs programs in this directory.

Like :$:, :PROG: refers to a different directory for each user ID. In Figure 1.4, the directory :SD:USER/1/PROG is the directory -- the Program Directory -- for user ID 1. The directory :SD:USER/65535/PROG is the Program Directory for WORLD.

The file R?LOGON is a file that runs when a user begins using a terminal.

PROGRAM DEVELOPMENT DIRECTORIES. The iRMX 86 PC Operating System is a program development environment in which language processors, editors, and other utilities are used to create, install, and run programs. Three empty directories exist on the iRMX 86 PC System diskette to support program development:

- Language Directory (:LANG:). This is a directory used to store language products, such as assemblers, compilers, and linkers. In the figure, the directory contains the Intel 8086 Assembler as file ASM86.

- Utility Directory (:UTILS:). This directory is used to store utilities, such as those shown as examples in Figure 2-3: the Intel Linker (LINK86), Locater (LOC86) and editor (EDIT).

- 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.

SYSTEM CONFIGURATION DIRECTORY (:CONFIG:)

This directory is used to inform the Operating System of the characteristics of your installation, for example, how many terminals are connected to the system. The configuration directory contains: the terminal definition file, and user definition files. These are described in Chapter 6, SYSTEM MANAGEMENT.

EXAMPLE: INSTALLING UTILITIES ON THE SYSTEM

This example shows how to get the Intel linker, locator, and editor onto your system disk. These utilities are available from Intel as part of the product iRMX 860. Assume for this example that you have inserted a copy of the iRMX 860 diskette into a second flexible disk drive.

Figure 2-7 shows how to attach the drive and get a directory listing of the files on the iRMX 860 diskette. The example following this one, Figure 2-8, shows how to copy the utilities that you want onto the system diskette.

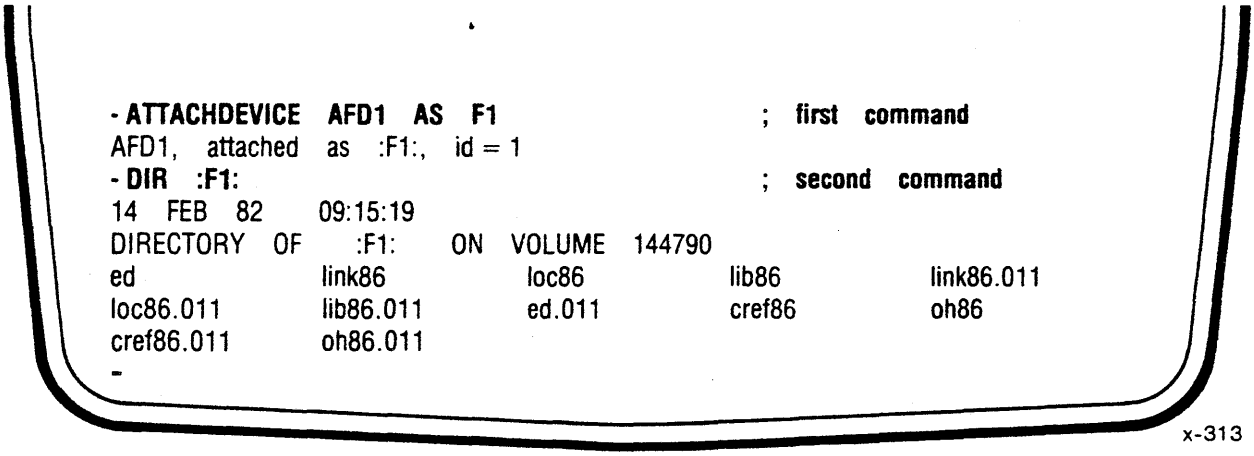


Figure 2-7. Installing Intel Utilities on System Disk

first command — This command allows the Operating System to use a second disk drive; it tells the system that you will be using the device AFD1, and that it will be referred to in subsequent commands as :F1:. The physical name (AFD1) is one of many names specified when the iRMX 86 PC System was configured by Intel. See the description of ATTACHDEVICE in Chapter 3 for a complete list of these physical names.

Because the device was attached by user ID 1 (confirmed on the line following the command), it can be detached only by user ID 1 or by the system manager (see the DETACHDEVICE command in Chapter 3).

second command — This command shows contents of the diskette except for the invisible files).

The next example screen, Figure 2-8, continues this sequence by copying selected utilities to the :UTILS: directory on the system disk.

```
-COPY :F1:* TO :UTILS:* QUERY                    ; third command
:F1:ed copy TO :UTILS:ed Y
:F1:ed copied TO :UTILS:ed
:F1:ed.011 copy TO :UTILS:ed.011? Y
:F1:ed.011 copied TO :UTILS:ed.011
:F1:link86 copy TO :UTILS:link86? Y
:F1:link86 copied TO :UTILS:link86
:F1:loc86 copy TO :UTILS:loc86? Y
:F1:loc86 copied TO :UTILS:loc86
:F1:lib86 copy TO :UTILS:lib86? N
:F1:link86.011 copy TO :UTILS:link86.011? Y
:F1:link86.011 copied TO :UTILS:link86.011
:F1:loc86.011 copy TO :UTILS:loc86.011? Y
:F1:loc86.011 copied TO :UTILS:loc86.011
:F1:cref86 copy TO :UTILS:cref86? E
-DIR :UTILS:                                     ; fourth command
14 FEB 82    09:17:17
DIRECTORY OF   :UTILS:    ON    VOLUME    145122
ed.011          link86.011      loc86.011      ed            link86
loc86

-
```

x-314

Figure 2-8. Copying Utilities to the System Disk

third command

This command uses two features that haven't been shown yet: wild cards, and copying with the QUERY parameter.

Wild Cards: The asterisks (*) in the inpath name and outpath name are wildcards. A later section explains wildcards in detail. Ignoring the effect of the QUERY parameter, the command says to, "Copy all files and directories from :F1: into the :UTILS: directory and give the copied files the same names."

Effect of QUERY: The QUERY parameter tells the Human Interface to prompt before copying a file. For example, the first prompt asks if the operator wishes to transfer the editor to the system disk, and the operator replied Y (Yes). The Operating System copies the file, confirms it, and then asks if the operator wants the file link86.011. The process continues. The file lib86 is passed over (N = No). When the operator replies with E (meaning Exit), the copy command terminates without copying any more files.

fourth command

The DIRectory listing shows that the files are in the :UTILS: directory. The editor, for example, is :UTILS:ed.

MORE ABOUT iRMX™ 86 COMMANDS

This section contains additional information about using iRMX 86 commands, and contains the following three sub-sections:

- ORDER OF DIRECTORY SEARCH BY THE HUMAN INTERFACE. Commands are simply program files, and the Human Interface will search certain directories to find the command.
- LINE EDITING CONTROLS. These are special characters that you can enter at the terminal to correct typing errors, and to control how the screen displays information.
- WILDCARDS. You can specify any of a group of files having similar names by means of special characters called wildcards.
- COMMAND LINE OPTIONS. When we described the general format of iRMX 86 commands, we did not tell you how to continue commands beyond one line, or how to designate that a special character can be used in a command line. You can do both.

EXAMPLE: COPYING THE UDI LIBRARIES TO THE SYSTEM DISK

You receive two diskettes with the iRMX 86 PC product: the System Diskette and an Include File Diskette. To use the UDI system calls, you will need two types of files from the Include File Diskette. You need one include file for each UDI system call that you use. And you need three interface libraries. Both types of files are described in the beginning of Chapter 4.

The Include File Diskette also contains include files and interface libraries for all iRMX 86 subsystems (Nucleus, Basic I/O System, Extended I/O System, Application Loader, and Human Interface).

If you use iRMX 86 system calls other than the UDI calls, you lose some of the advantages of the UDI. And of course you need the reference manuals that describe the system calls. The manuals are described in Chapter 7. The system calls for iRMX 86 subsystems are listed in Appendix B.

Figure 2-9 shows how to transfer the UDI files from the Include File Diskette to the system disk. The steps shown in the example are:

1. Creating a directory on the system disk for the UDI files.
2. Copying the include files to the system disk.
3. Copying the interface libraries to the system disk.
4. Displaying the contents of a typical include file (for the UDI system call DQ$ALLOCATE.

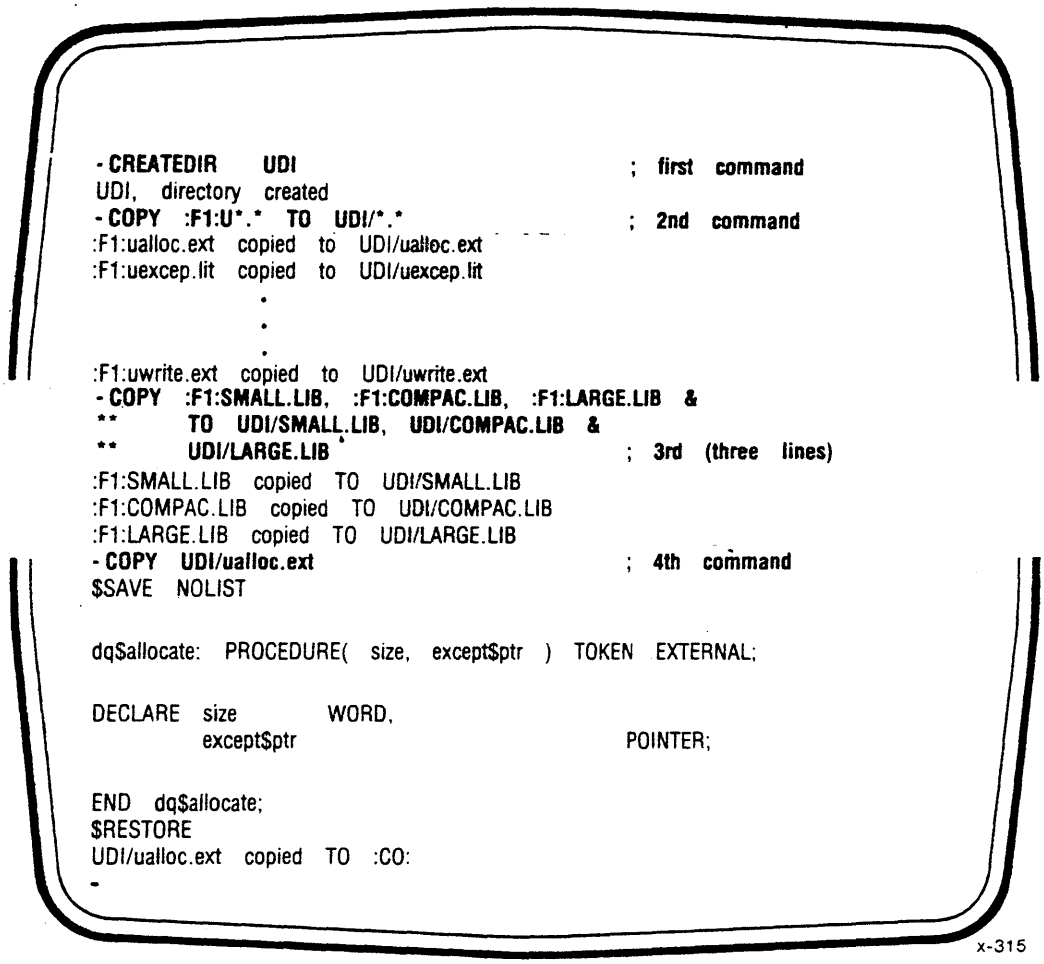


Figure 2-9. Transferring UDI files to System Disk

first command — This creates a directory on the system disk for the UDI files. Because the default directory is still :SD:, the directory pathname is :SD:UDI.

2nd command — Because of the wild cards asterisks, this single command transfers every UDI include file to the system disk. Because of how many files are transferred (one per UDI system call), the figure indicates most of the transfer with vertical dots.

3rd command — This command transfers the three UDI interface libraries, and also shows how you can continue commands beyond one line by typing an ampersand before the RETURN. This and other line editing features are described in a later section.

4th command — A typical include file -- for DQ$ALLOCATE -- is displayed on the screen. Note that TO :CO: doesn't have to be typed, it is the default preposition and output pathname for COPY.

ORDER OF DIRECTORY SEARCH BY THE HUMAN INTERFACE

When you enter a command name, you can enter the complete pathname of the command or you can enter just the last component of the pathname.

- If you enter the complete pathname of the command (that is, if you include a logical name as the prefix portion of the pathname), the Operating System searches only the device and directory you specify for the command. If it cannot find the command there, it returns an error message.

- If you enter only the last component of the pathname (such as COPY instead of :F1:SYSTEM/COPY), the Operating System automatically searches certain directories for the command. It does not return an error message unless it has searched each of the directories without finding the command file. The Operating System searches the following directories, in order, for commands:

  ```
  :$:
  :PROG:
  :SYSTEM:
  :LANG:
  :UTILS:
  ```

When writing your own commands, you can take advantage of the order in which the Operating System searches directories. For example, suppose you write your own copy command, one that provides more or different functions than the Human Interface COPY command. If you want to invoke your program whenever you type COPY, you can place your file called COPY in your default directory (:$:). Because the Operating System searches the default directory before searching the :SYSTEM: directory (which contains the Human Interface COPY command), it will invoke your copy program.

If you still want to be able to invoke the Human Interface COPY command, you can do so by entering: :SYSTEM:COPY.

EXAMPLE: CREATING A PRIVATE DISK

The following example, Figure 2-10, shows how to prepare a disk that one programmer uses for developing programs. The example shows how to:

1. Format the disk (we assume it is a flexible diskette, and use the default FORMAT parameters).

2. Create a directory on the disk named PROG.

3. Create a directory within the new PROG directory named TEST.

4. Make the directory PROG the default directory.

5. Use the DIR command to confirm that everything worked.

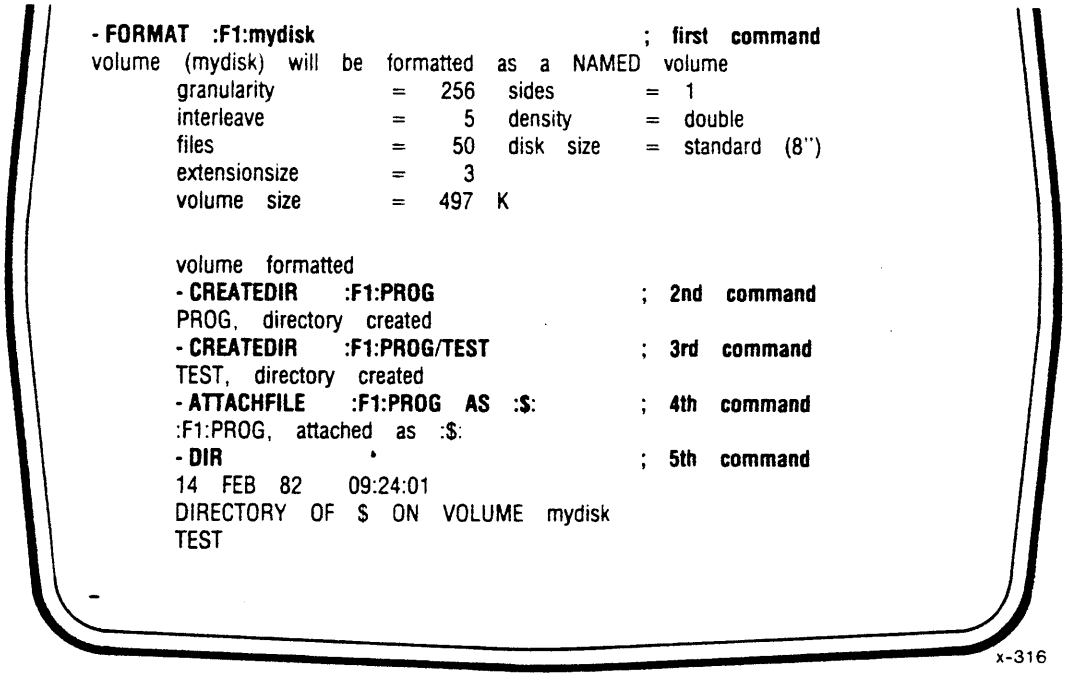


Figure 2-10. Creating A Private Disk

first command — This shows how to format a diskette. Remember that the second flexible disk drive on the system was attached as :F1: in a previous example.

When you format a disk or other mass storage device, the system destroys any data that was on the device and writes onto the disk information needed by the iRMX 86 file system, including the root directory and three invisible "bitmap" files that were shown in a previous DIR example. In Chapter 6, we show how to format flexible diskettes and Winchester disks.

The system displays a message describing the format characteristics. This example uses the default values (like interleave = 5), but you can specify other values by means of parameters to the FORMAT command.

second command — This command creates a directory named PROG on the newly formatted disk.

third command — This creates a directory named TEST in the new directory PROG.

fourth command — The PROG directory is attached as this user ID's default directory.

fifth command — A directory listing confirms that the PROG directory is the default directory, and that it contains the directory TEST.

LINE EDITING CONTROLS

The examples shown in this chapter imply that no typing errors occur. This is unrealistic, and the iRMX 86 Operating System provides extensive line editing controls to allowing you to correct typing mistakes. Line editing controls also include a variety of other capabilities, including control over how the system sends output to your console terminal. These functions work nearly any time you are entering text at a terminal (for example, while using the Intel editor).

This section describes how to use each control feature. The best way to understand line editing controls is to try them at a terminal.

Controlling Input to a Terminal

You can use several characters to control and edit terminal input. Some of these characters correspond to single keys on your terminal (such as carriage return or rubout). For others, called control characters, you must press the CTRL key, and while holding it down, also press an alphabetical key. This manual designates control characters as follows:

CTRL/character

The editing and control characters for terminal input are:

CARRIAGE RETURN or LINE FEED	Terminates the current line and positions the cursor at the beginning of the next line. Entering either of these characters adds a carriage return/line feed pair to the input line.
RUBOUT	Deletes (or rubs out) the previous character in the input line. Each RUBOUT removes a character from the screen and moves the cursor back to that character position.
CTRL/r	If the current input line is not empty, this character reprints the line with editing already performed. This enables you to see the effects of the editing characters entered since the most recent line terminator. If the current line is empty, this character reprints the previous line, up to the point of the line terminator. Additional CTRL/r characters display previous lines until there are no more lines that have been saved. Subsequent CTRL/r characters display the last line found. Trying the feature is the best way to understand how it works.
CTRL/u	Discards the current line and the entire contents of the type-ahead buffer.
CTRL/x	Discards the current input line. This character echoes the "#" character, followed by a carriage return/line feed, at the terminal.

Controlling Output to a Terminal

Output to a terminal operates in one of four modes. You can switch the current output mode dynamically to any of the other output modes by entering output control characters. The output modes and their characteristics are as follows:

Normal	The Terminal Support Code accepts output from the application system and immediately passes the output to the terminal for display.
Stopped	The Terminal Support Code accepts output from the application system, but it queues the output rather than immediately passing it to the terminal.
Scrolling	The Terminal Support Code accepts output from the application system, and it queues the output as in the stopped mode. However, rather than completely preventing output from reaching the terminal, it sends a predetermined number of lines (called the scrolling count) to the terminal whenever the operator enters a control character at the terminal.
Discarding	The Terminal Support Code discards output from the application system without displaying or queuing the output.

The following control characters, when entered at the terminal, change the output mode for the terminal.

CTRL/o	Places the terminal in discarding mode if the terminal is in a mode other than discarding mode. If the terminal is already in discarding mode, the CTRL/o character returns the terminal to its previous output mode.
CTRL/q	Resumes previous output mode. If you enter this character after stopping output with the CTRL/s character, output continues in the same manner as before you entered the CTRL/s (that is, if your terminal was in scrolling mode before you entered CTRL/s, output resumes in scrolling mode). Entering CTRL/q at any other time places your terminal in normal mode (that is, all output is displayed at the terminal without waiting for permission to continue). Therefore, you can use CTRL/q to reverse the effect of a CTRL/w and get your terminal out of scrolling mode.
CTRL/s	Places the terminal in stopped mode (stops output). You can resume output without loss of data by entering the CTRL/q character. If the terminal is in discarding mode (as a result of a CTRL/o character), the CTRL/s character has no effect on output.

CTRL/t — Places the terminal in scrolling mode and sets the scroll count to one. This means that you must enter another CTRL/t character after each displayed line in order to continue the display.

CTRL/w — Places the terminal in scrolling mode. In this mode, the terminal displays output several lines at a time (usually, enough lines to fill the screen) and then waits for user input to continue. When you enter another CTRL/w character, the terminal displays the next screen of information. The scrolling count is selectable; refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information.

Entering the CTRL/w character while the terminal is already in scrolling mode increments the scrolling count by the original scrolling count value. Therefore, you can use CTRL/w to increase the number of lines the terminal displays before stopping. Entering an input line resets the scroll count to its original value.

The following two control characters can affect output to the terminal:

CTRL/c — Deletes the type-ahead buffer and causes the Operating System to abort the currently-executing program. If you enter a Human Interface command to initiate a program, you can enter CTRL/c to stop it.

CTRL/z — If typed in response to the Human Interface prompt, this will re-initialize the interactive job for the terminal (as described in Chapter 2, in the section SETTING THE DATE AND TIME).

Type-Ahead

Sometimes a person will type faster than the iRMX 86 Operating System can process the input. Because of type-ahead, commands and data typed ahead of processing will not be lost. (Characters typed ahead are not echoed on the screen.) The Operating System starts processing the first line, and saves additional lines in a type-ahead buffer. It reads subsequent lines from the type ahead buffer. If the type-ahead buffer becomes full, the Terminal Support Code sounds the terminal bell.

Escape Sequences

The control characters listed are appropriate for all but the most unusual applications. But if you wish to re-define any of these characters (for example, to set the number of lines that the screen scrolls), you can do so with escape sequences. Refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for detailed information.

WILD CARDS

Wild cards provide a shorthand notation for specifying several files in a single reference when entering commands. You can use either of two special wild card characters in the last component of a pathname to replace some or all characters in that component.

The wild card characters are:

? The question mark matches any single character. The Human Interface allows any character to appear in that character position. It selects every file that meets this requirement. For example, the name "FILE?" could imply all of the following files:

```
FILE1   FILE2       FILEA
```

* The asterisk matches any number of characters (including zero characters). The Human Interface allows any number of characters to appear in that character position. It selects every file that meets this requirement. For example, the name "FILE*" could imply all of the following files:

```
FILE1
FILE.OBJ
FILE
FILECHANGE
```

You can use multiple wild cards in a single pathname. For example, the name:

```
?PIF?.*
```

matches every file whose second through fourth characters are "PIF" and whose sixth character is a period. These files could include all of the following names (or more):

```
RPIFC.LIB
EPIFL.TXT
HPIFC.
```

You can use wild cards in both input pathnames (files that commands read for information) and output pathnames (files into which commands write information). For example, in the command:

```
COPY A* TO B*
```

the A* represents the input pathname and B* represents the output pathname. The Human Interface searches the appropriate directory for all files that begin with the "A" character. Then it copies each file to a file of the same name, but beginning with the "B" character.

Figures 2-8 and 2-9 show examples of using wild cards to copy multiple files.

Be aware, when using wild cards, that:

- Wild cards are valid in the last component of the pathname only. Therefore, :F1:SYSTEM/APP1/FILE* is a valid pathname, but :F1:SYSTEM/APP*/FILE1 is not valid.

- You can negate the meaning of a wild card character by enclosing it in quotes, either single (') or double ("). See LINE EDITING earlier in this section.

- When you specify input and output pathnames in commands, you can specify lists of pathnames, separated by commas. For example:

  ```
  COPY A,B,C TO D,E,F
  ```

  copies A to D, B to E, and C to F. If you use a wild cards in any one of the output pathnames, you must use the same wild cards in the same order in the corresponding input pathname. The term "same order" means that if you use both the "*" and the "?" characters, their ordering must be the same in both the input and output pathnames. For example, the following is valid:

  ```
  COPY A*B?C* TO *DE?FGH*I
  ```

  However, the following is not valid because the wild cards are out of order:

  ```
  COPY A*B?C* TO *DE*FGH?I
  ```

- If you use wild cards in an input pathname, you can omit all wild cards from the corresponding output pathname to cause the Human Interface to perform file concatenation. For example, suppose a directory contains files A1, B1, and C1. The following command is valid:

  ```
  COPY *1 TO X
  ```

  It copies files in the following manner:

  ```
  A1 TO X
  B1 AFTER X
  C1 AFTER X
  ```

- The "*" character matches as close to the end of the pathname as possible. For example, suppose the directory contains the file "ABXCDEFXGH", and you enter the command:

  ```
  COPY *X* TO *1*
  ```

  This command copies:

  ```
  ABXCDEFXGH TO ABXCDEF1GH
  ```

  The first asterisk matches the characters "ABXCDEF", and the second asterisk matches the characters "GH".

COMMAND LINE OPTIONS

In entering commands from a terminal, you may occasionally need to know about the following command line options.

Commands That Require More Than One Line

An ampersand character (&) indicates that the command continues on the next line. When you include the ampersand character, the Human Interface displays two asterisks (**) on the next line to prompt for the continuation line. All characters appearing after the continuation mark but before the line terminator are interpreted as comments. After you enter the line terminator without a preceding ampersand character, the invoked command receives the entire command string as a single command.

Although the Human Interface places no restriction on the number of characters in a command, each terminal line can have a maximum of 255 characters, including any punctuation, embedded blanks, continuation mark, non-executable comments, and carriage return. If your command requires more characters, use continuation lines. Within available memory limits, you can use as many continuation lines for a given command as you desire.

Quoting Characters in a Command

Two single-quote (') or double-quote (") characters remove the semantics of special characters they surround. For example, if you surround an ampersand character (&) with single quotes, the ampersand is not recognized as a continuation character. The same holds for other characters such as asterisk (*), question mark (?), equals (=), semicolon (;), and others. The only special characters not affected by the quoting characters are the pathname separators (/ and ^), semicolon (:), and dollar sign ($). Although you can use either single quotes or double quotes as quoting characters, you must use the same quoting character at the beginning and at the end of your quoted string. To include the quoting character inside a quoted string, you can either specify the character twice, or use the other quoting character. For example:

```
'can''t' or     "can't"
```

Prepositions and Path Lists

Earlier we showed the general form of iRMX 86 commands:

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

This section contains more information about prepositions, and about inpath-lists and outpath-lists.

PREPOSITIONS. Preposition parameters in a command line tell the command how you want it to process the output file or files. The Human Interface commands usually provide three options in the choice of a preposition: TO, OVER, and AFTER. The preposition AS is also available for use in the ATTACHDEVICE and ATTACHFILE commands. The TO preposition and :CO: (console screen) will be used by default if you do not specify a preposition and an output file.

The prepositions have the following meaning:

TO — Causes the command to send the processed output to new files; that is, to files that do not already exist in the given directory. If a listed output file already exists, the command displays the following query at the console screen:

```
<pathname>, already exists, OVERWRITE?
```

Enter a Y or y if you wish to write over the existing file. Enter any other character if you do not wish the file to be overwritten. In the latter case, the command does not process the corresponding input file but rather goes to the next input file in the command line. Commands process input files and write to output files on a one-for-one basis. For example:

```
COPY A,B TO C,D
```

copies file A to file C and file B to file D.

OVER — Causes the command to write your input files to the output files in sequence, destroying any information currently contained in the output files. It creates new output files if they do not exist already. 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.

AFTER — Causes the command to append the contents of one or more files to the end of one or more new or existing files (file concatenation). For example:

```
COPY IN1,IN2 AFTER DEST1,DEST2
```

appends the contents of file IN1 to the the end of file DEST1, and appends the contents of IN2 to the end of DEST2.

AS — A special preposition used with the ATTACHDEVICE and ATTACHFILE commands. When you use the AS preposition, the Operating System does not assume that the command contains input pathnames and output pathnames. Rather, it sees the parameters as entities that it must associate (for example, ATTACHFILE associates a pathname with a logical name).

Inpath-List and Outpath-List

An inpath-list specifies the files on which a command is to operate. An outpath-list defines the destination or destinations of the processed output. Each inpath-list or outpath-list consists of a pathname (or logical name) or list of pathnames. If you specify multiple pathnames, you must separate the individual pathnames with commas. Embedded blanks between pathnames are optional. You can also use wild cards to indicate multiple pathnames (refer to the "Wild Cards" section of this chapter).

Usually when you specify multiple pathnames, each pathname in the inpath-list has a corresponding pathname in the outpath-list. For example, the command:

```
COPY A, B  TO  C, D
```

copies file A to file C and also copies file B to file D. Therefore, A and C are corresponding pathnames, and so are B and D. However, there are some instances when the number of input pathnames you enter differs from the number of output pathnames. The validity of the operation depends on whether the pathname lists contain single pathnames, lists of pathnames, a wild-card pathname, or lists of wild-card pathnames. Table 2-1 lists the possibilities and describes the Human Interface's action in each instance. The following sections discuss the Human Interface's actions in more detail.

Table 2-1. Input Pathname and Output Pathname Combinations

Inpath-list	Outpath-list	Human Interface Action
single pathname	single pathname	one-for-one match
single pathname	list of pathnames	error
single pathname	wild-card pathname	error
single pathname	list of wild cards	error
list of pathnames	single pathname	concatenate
list of pathnames	list of pathnames	one-for-one match
list of pathnames	wild-card pathname	error
list of pathnames	list of wild cards	error
wild-card pathname	single pathname	concatenate
wild-card pathname	list of pathnames	error
wild-card pathname	wild-card pathname	one-for-one match
wild-card pathname	list of wild cards	error
list of wild cards	single pathname	concatenate
list of wild cards	list of pathnames	concatenate
list of wild cards	wild-card pathname	concatenate
list of wild cards	list of wild cards	one-for-one match

ONE-FOR-ONE MATCH. The combinations in Table 2-1 that are marked "one-for-one match" are those in which each element in the inpath-list is matched with an element of the outpath-list. An example of this is the command:

```
COPY A*, B*  TO  C*, D*
```

In this case, the Human Interface copies all files beginning with the character "A" to corresponding files beginning with the character "C". When it finishes this operation, it advances past the comma to the next set of pathnames (copies all files beginning with "B" to corresponding files beginning with "D").

CONCATENATE. The combinations in Table 2-1 that are marked "concatenate" are those in which there are multiple input pathnames that correspond to a single output pathname. In this situation, the Operating System automatically appends the remaining input files to the end of the specified output file, regardless of the preposition you specify.

This allows you to combine one-for-one file operations (as in TO or OVER preposition) with file concatenation (as in the AFTER preposition) in a single command, and thus avoid entering an extra command to perform a separate concatenation operation. For example:

```
COPY A,B,C  TO  D
```

copies file "A" to file "D" and appends files "B" and "C" to the end of file "D."

Notice that this concatenation occurs only when there are multiple elements in the inpath-list that correspond to a single element of the outpath-list. This means that the following commands are invalid:

```
COPY A, B, C     TO  D, E      ; INVALID COMMAND

COPY A*, B*, C*  TO  D*, E*    ; INVALID COMMAND
```

ERROR CONDITIONS. The combinations in Table 2-1 that are marked "error" indicate invalid operations. For these combinations, the Human Interface returns an error message without performing the requested operation.

***

CHAPTER 3. HUMAN INTERFACE COMMANDS

The commands described in this chapter are supplied by Intel with the Preconfigured iRMX 86 Operating System. You can use these commands to perform a number of highly convenient file management and system functions. When you invoke a command,

1. You type the command name and parameters (e.g., "COPY FIRST TO SECOND").

2. The Operating System loads the appropriate command file (for example, :SD:SYSTEM/COPY) and executes the program the way that you specify in the command line.

The bulk of this chapter contains descriptions, arranged alphabetically, of each Human Interface command.

If you are a new user of the Human Interface, we suggest that you review the information on iRMX 86 commands and files in Chapter 2.

Human Interface commands are program files in the SYSTEM directory. When you type a command on the terminal, the Operating System looks for the file in a series of directories. The directories, listed in the order they are searched by the Human Interface, are:

```
:$:
:PROG:
:SYSTEM:
:LANG:
:UTILS:
```

You can place commands in any directory that the Human Interface automatically searches, and invoke the command with its name. You can also invoke commands that are not in directories searched by the Human Interface.

HUMAN INTERFACE COMMAND DICTIONARY

The Human Interface Command Dictionary, Table 3-1, briefly describes each command and gives its page number. The Dictionary divides the commands into functional groups:

File management commands

Volume management commands

Multi-access commands

General utility commands

Table 3-1. Human Interface Command Dictionary

Command	Synopsis	Page
	File Management Commands	
ATTACHFILE	Associates a logical name with an existing file.	3-13
COPY	Creates new data files, or copies files to other pathnames.	3-24
CREATEDIR	Creates one or more new directories.	3-28
DELETE	Deletes data files and empty directories from a volume on secondary storage.	3-33
DETACHFILE	Removes the association of a logical name with a file.	3-38
DIR	Lists a directory's filenames (and optionally, file attributes).	3-40
DOWNCOPY	Copies files and directories from an iRMX 86 volume mounted on a secondary storage device to an ISIS-II secondary storage device.	3-53
PERMIT	Grants or rescinds user access to a file.	3-69
RENAME	Renames files or directories.	3-74
UPCOPY	Copies files and directories from an ISIS-II secondary storage device to an iRMX 86 volume mounted on a secondary storage device.	3-92
	Volume Management Commands	
ATTACHDEVICE	Attaches a new physical device to the system and associates its physical name with a logical name.	3-7
BACKUP	Copies named files to a backup volume.	3-16
DETACHDEVICE	Removes a physical device from system use and deletes its logical name.	3-35
DISKVERIFY	Verifies the data structures of named and physical volumes.	3-48

Table 3-1. Human Interface Command Dictionary (continued)

Command	Synopsis	Page
	Volume Management Commands (continued)	
FORMAT	Formats an iRMX 86 volume.	3-56
RESTORE	Copies files from a backup volume to a named volume.	3-77
	Multi-Access Commands	
INITSTATUS	Displays the initialization status of Human Interface terminals.	3-63
JOBDELETE	Deletes a running interactive job.	3-65
LOCK	Prevents the Human Interface from automatically creating an interactive job after the job has been deleted.	3-67
SUPER	Changes the operator's user ID into that of the system manager (user ID 0) and grants the ability to change to other user IDs.	3-87
	General Utility Commands	
DATE	Sets or resets the system date, or displays the current date and time.	3-29
DEBUG	Transfers control to the iSBC 957B package to debug an iRMX 86 application program.	3-31
SUBMIT	Reads, loads, and executes a string of commands from secondary storage instead of the keyboard.	3-83
TIME	Sets or resets the system clock, or displays the current system date and time.	3-90
VERSION	Displays the version number of command programs.	3-95

ERROR MESSAGES

Each command can generate a number of error messages which indicate errors in the way you specified the command. The messages that apply to a specific command are listed with that command. However, the following are general error messages that can appear with many of the commands:

- command not found

  There is no file whose pathname is the same as the command name you specified, nor can the Human Interface find the file in any of the directories it automatically searches.

- <logical name>, device does not belong to you

  The device you specified was originally attached by a user other than WORLD or you.

- <pathname>, file does not exist

  The pathname you specified does not represent an existing file.

- <pathname>, invalid file type

  You specified a data file for an operation that required a directory, or vice versa.

- <logical name>, invalid logical name

  The logical name you specified contains unmatched colons, is longer than 12 characters, or contains invalid characters.

- <pathname>, invalid pathname

  The pathname you specified contains invalid characters or a component of the pathname (other than the last one) does not exist or does not represent a directory.

- <logical name>, is not a device connection

  The logical name you specified does not represent a connection to a physical device.

- <logical name>, logical name does not exist

  The logical name you specified does not exist.

- parameters required

  The command you specified cannot be entered without parameters.

- program version incompatible with system

  The command and the Operating System are not compatible. The command expects to obtain information from internal tables that are not present. Therefore the command cannot run successfully.

- <control>, unrecognized control

  The control you entered is not valid for the specified command.

- <exception value> : <exception mnemonic> while loading command

  The Operating System encountered an exceptional condition while attempting to load the command into memory from secondary storage. The message lists the exception code encountered.

- <exception value> : <exception mnemonic>

  An operational error occurred during the execution of the command. The <exception value> and <exception mnemonic> portions of the message indicate the exception code encountered.

- <parameter>, <exception value> : <exception mnemonic>

  The command encountered an exceptional condition while attempting to process the <parameter> portion of the command. The <exception value> and <exception mnemonic> portions of the message indicate the exception code encountered.

COMMAND SYNTAX SCHEMATICS

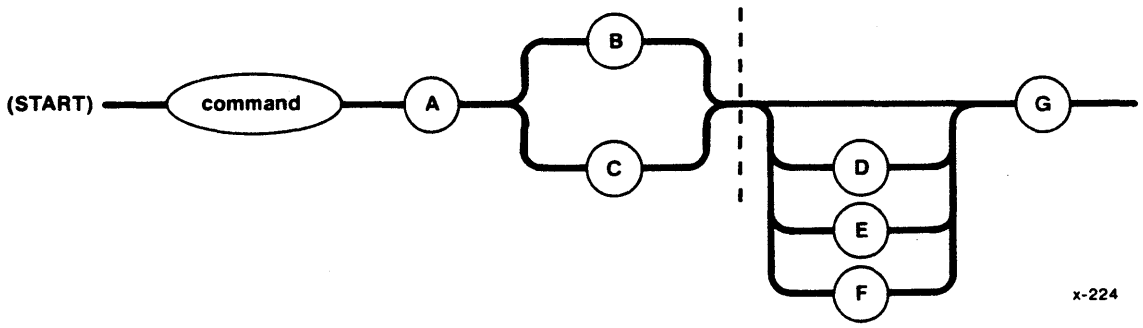
The syntax for each command described in this chapter is presented by means of a "railroad track" schematic, with syntactic elements scattered along the track. Your entrance to any given schematic is always from left to right, beginning with some command name entry.

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; the Human Interface makes no distinction between cases in alphabetic characters. Syntactic elements shown in lowercase characters are generic terms, which means that you supply the specific item, such as the pathname for a file.

The vertical dotted line separates the position-dependent parameters from those that are position-independent. Parameters to the left of the dotted line must be entered in the order listed (from left to right). Parameters to the right of the dotted line can be entered in any order (as long as they obey the rest of the syntax).

The example that follows shows all 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. In still other cases, the main track itself diverges into two separate tracks, which means that you must select one parameter or the other but not both.

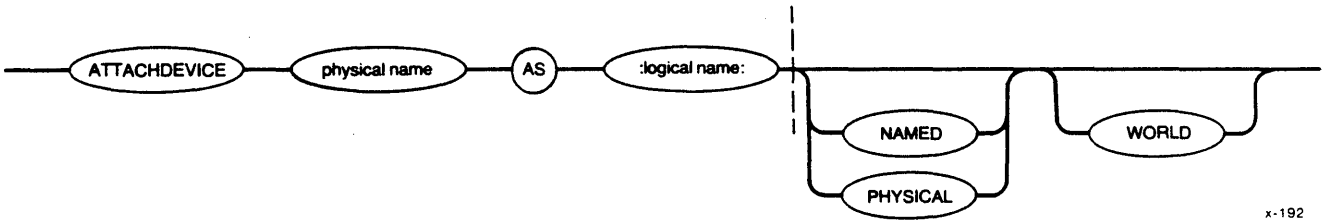


In this example:

- A is a required element. It is position-dependent; it must be entered first.
- Either B or C is required but not both. These elements are also position-dependent. Whichever element you enter must follow A immediately.
- D, E, or F are all optional but only one can be selected. These are position-independent elements. If you select one of these elements, you can enter it before or after G.
- G is required. It is a position-independent parameter. You can enter it before or after D, E, or F.

ATTACHDEVICE

This command attaches a physical device to the Operating System, associates a logical name with the device, and makes the logical name accessible to all users. The logical name is used in all other commands to refer to the device. The format of the command is as follows:



INPUT PARAMETERS

physical name	Physical device name of the device to be attached to the system. This name must be one of the names in Table 3-2.
AS	Preposition; required for the command.
:logical name:	A 1- to 12-character name, that represents the logical name to be associated with the device. Colons surrounding the logical name are optional; however, if you use colons, you must use matching colons. After ATTACHDEVICE attaches and catalogs the device, any command you enter or program code you run must specify the logical name in order to access the device
NAMED	Specifies that the volume mounted on the device is already formatted for NAMED files. Examples of volumes that can contain named files 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 considered to be a single, large file. Examples include line printers and terminals. See the FORMAT command in this chapter for a further description of PHYSICAL volumes.

WORLD	Specifies that user ID WORLD (65535 decimal) is the owner of the device. This implies that any user can detach the device. If you omit this parameter, your user ID is listed as the owner of the device. In this case, only you and the system manager can detach the device.

DESCRIPTION

ATTACHDEVICE attaches a device to the system and associates a logical name for the device. The logical name is the means by which all users can access the device.

Devices must have their characteristics defined at configuration time before they can be attached with the ATTACHDEVICE command. Table 3-2 lists the physical device names available for the Preconfigured iRMX 86 Operating System.

One frequent use of the ATTACHDEVICE command is to attach a new device, such as a new disk drive or a line printer, without having to reconfigure portions of the Operating System. (See the DETACHDEVICE command in this chapter for a description of how to detach a device from the system without reconfiguring.)

Unless you have a user ID of WORLD (65535) or specify the WORLD parameter, once you attach a device, only you and the system manager can detach the device. This prevents users from detaching devices belonging to other users and prevents you from accidentally detaching system volumes. However, if you have a user ID of WORLD or specify the WORLD parameter, any device that you attach can be detached by any other user. Refer to the DETACHDEVICE command for more information.

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

<physical name>, attached as <logical name>, id = <user id>

where <physical name> and <logical name> are as specified in the ATTACHDEVICE command and <user id> is your user ID (or WORLD, if you specify the WORLD parameter).

Table 3-2. Physical Device Names for the iRMX™ 86 PC System

8-inch Flexible Disk Drives					
Device Name	iSBC® Controller	Device Type	Unit Number	Sides/ Density	Bytes-per Sector
AF0	208	Shugart SA800	0	1/Single	128
AF1	208	Shugart SA800	1	1/Single	128
AFD0	208	Shugart SA800	0	1/Double	256
AFD1	208	Shugart SA800	1	1/Double	256
AFD2	208	Shugart SA800	2	1/Double	256
AFD3	208	Shugart SA800	3	1/Double	256
AFDD0	208	Shugart SA850	0	2/Double	256
AFDD1	208	Shugart SA850	1	2/Double	256
AFDX0	208	Shugart SA850	0	2/Double	1024
AFDX1	208	Shugart SA850	1	2/Double	1024
WF0	218	Shugart SA800	0	1/Single	128
WF1	218	Shugart SA800	1	1/Single	128
WFD0	218	Shugart SA800	0	1/Double	256
WFD1	218	Shugart SA800	1	1/Double	256
WFD2	218	Shugart SA800	2	1/Double	256
WFD3	218	Shugart SA800	3	1/Double	256
WFDD0	218	Shugart SA850	0	2/Double	256
WFDD1	218	Shugart SA850	1	2/Double	256
WFDX0	218	Shugart SA850	0	2/Double	1024
WFDX1	218	Shugart SA850	1	2/Double	1024
5 1/4-inch Flexible Disk Drives					
Device Name	iSBC® Controller	Device Type	Unit Number	Sides/ Density	Bytes-per Sector
AMFD0	208	Shugart SA450	0	2/Double	256
AMFD1	208	Shugart SA450	1	2/Double	256
AMFD2	208	Shugart SA450	2	2/Double	256
AMFD3	208	Shugart SA450	3	2/Double	256
AMFDD0	208	Shugart SA460	0	2/Double	512
AMFDD1	208	Shugart SA460	1	2/Double	512
WMD0	218	Shugart SA450	0	2/Double	256
WMD1	218	Shugart SA450	1	2/Double	256
WMD2	218	Shugart SA450	2	2/Double	256
WMD3	218	Shugart SA450	3	2/Double	256
WMDD0	218	Shugart SA460	0	2/Double	512
WMDD1	218	Shugart SA460	1	2/Double	512

Table 3-2. Physical Device Names for the iRMX™ 86 PC System (continued)

Winchester Disk Drives					
Device Name	iSBC Controller	Device Type	Unit Number	Sides/ Density	Bytes-per Sector
IW0	215	Priam 3450			1024
MW0	215	Memorex 101			1024
PW0	215	Pertec D8000			1024
SW0	215	Shugart SA1002/1004			1024

Other Devices	
BB	Byte bucket
STREAM	Stream file device
T0	Terminal connected to Single Board Computer
T1-T4	Terminals connected to iSBC 534 Ports, 0-3 respectively
LP	Line Printer

ERROR MESSAGES

- <device name>, cannot be ATTACHED as <type>

  The device specified by <device name> cannot support the type of files specified by <type> (NAMED or PHYSICAL). ATTACHDEVICE does not attach the device. For example, the NAMED option is not valid for a device such as a line printer.

- <device name>, device already attached

  The specified device has already been attached. ATTACHDEVICE does not attach the device.

- <device name>, device does not exist

  The physical device name you specified does not correspond to a name the Operating System recognizes. That is, the name is not in Table 3-2. ATTACHDEVICE does not attach the device.

- <logical name>, logical name already exists

  The specified logical name is already defined for some other device, or for a file. ATTACHDEVICE does not attach the device.

- 0085 : E$LIST, 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. You can use the device after formatting the volume as a named volume or after inserting a named volume in the device.

- <logical name>, volume not formatted
  <logical name>, <exception value> : <exception mnemonic>

  ATTACHDEVICE attempted to attach a device as a named device and encountered an I/O error while searching for the volume's root directory. This usually indicates that the volume is not formatted. However, ATTACHDEVICE does attach the device.

- <logical name>, volume not mounted

  The specified device does not contain a volume. However, ATTACHDEVICE does attach the device.

- <exception value> : <exception mnemonic>, while collecting device name

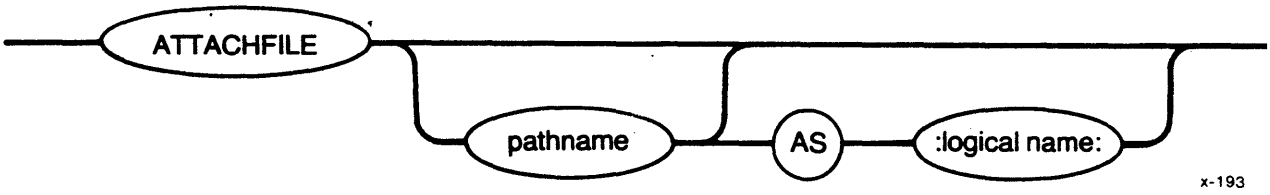
  ATTACHDEVICE encountered an exceptional condition while searching for the device name in the tables maintained by the Basic I/O System. This message lists the resulting exception code.

- <exception value> : <exception mnemonic>, while collecting logical name

  ATTACHDEVICE encountered an exceptional condition while attempting to assign the logical name to the device. This message lists the resulting exception code.

ATTACHFILE

This command allows you to associate a logical name with an existing file. The format of this command is as follows:



INPUT PARAMETERS

pathname	Pathname of the file to which the Human Interface associates a logical name.
:logical name:	1- to 12-character name that represents the logical name to be associated with the file. Colons surrounding the logical name are optional; however, if you use colons, you must use matching colons. If you omit this parameter, the default logical name is :$:.

If you enter the ATTACHFILE command without parameters, the default is:

```
ATTACHFILE :HOME: AS :$:
```

DESCRIPTION

The ATTACHFILE command allows you to associate a logical name with an existing file. After making this association, you can use the logical name, instead of the entire pathname, to refer to the file.

When the attachment is complete, ATTACHFILE displays the following message:

<pathname>, attached AS <logical name>

where <pathname> and <logical name> are as specified in the ATTACHFILE command.

ATTACHFILE makes the association between a file and a logical name, and makes the name known to any program that you run at your terminal. If another file is known by the logical name, ATTACHFILE deletes the previous association in order make the new one.

The logical name is known only within your interactive job. Therefore, several users can specify the same logical name without affecting each other.

If you specify a pathname for a file but omit the logical name, ATTACHFILE attaches the file as :$:. This allows you to change your default prefix. Changing your default prefix can be useful when you want to manipulate files that reside in a directory other than the one specified by your original default prefix. For example, suppose you have a file that you normally refer to as:

```
:PROG:SOURCE/PLM/INTERRUPT/TEST.P86
```

You can change your default prefix with the command:

```
ATTACHFILE :PROG:SOURCE/PLM/INTERRUPT
```

Then, you can refer to the file as simply:

```
TEST.P86
```

When you finish using the files in directory :PROG:SOURCE/PLM/INTERRUPT, you can return your default prefix to its original setting by entering:

```
ATTACHFILE
```

This is the same as entering:

```
ATTACHFILE :HOME: AS :$:
```

:HOME: is a logical name that refers to the same directory as your original default prefix. Therefore, you can change your default prefix as much as you like with ATTACHFILE and return to the original setting by making reference to :HOME:. However, you cannot use ATTACHFILE to change the meaning of :HOME:. (Also, you cannot use ATTACHFILE to change the meaning of :CI: and :CO:.)

The logical name created with ATTACHFILE remains valid until one of the following situations occur:

- A DETACHFILE command (described later in this chapter) dissolves the association between file and logical name.
- The interactive session that specified the ATTACHFILE command terminates processing. This occurs when a user, in response to the Human Interface prompt, enters a Control-Z character to reinitialize the interactive job. In this case, the Operating System deletes the interactive job and then recreates it. This restores the interactive job to its initial state.
- A task deletes the connection to the file via a Basic I/O System or Extended I/O System call (refer to Appendix B for descriptions of the iRMX 86 I/O Systems).
- A user forcibly detaches the volume containing the file via the DETACHDEVICE command (described later in this chapter).

ERROR MESSAGES

- <pathname>, list of logical names not allowed

  You entered more than one logical name as input to ATTACHFILE.

- <logical name>, list of pathnames not allowed

  You entered more than one pathname as input to ATTACHFILE.

- <logical name>, logical name not allowed

  You attempted to attach a file using a logical name :HOME:, :CI:, or :CO:. You cannot change the meaning of these logical names.

- <logical name>, not a file connection

  The logical name you specified, <logical name>, is already cataloged in object directory of the session and does not represent a file.

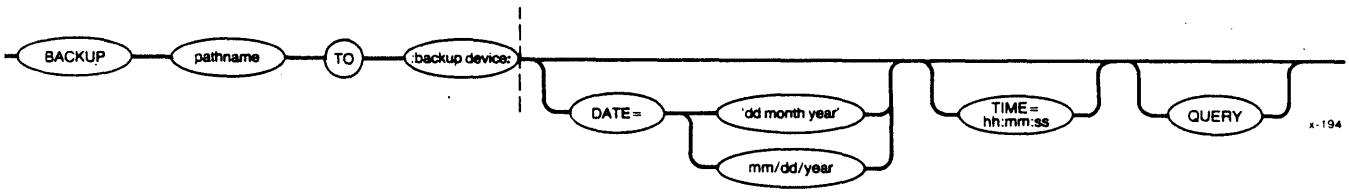
- <logical name>, too many logical names

  ATTACHFILE is unable to catalog the file's name in the object directory because an internal Operating System table is full.

BACKUP

This command saves files from 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 all files in the volume, beginning with the root directory.

'dd month year' — One form of the 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. If you use this form of the date parameter, 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.
- month — Designation for the month. You can enter the whole name (such as AUGUST) or enough characters to distinguish one month from another (for example, AU, to distinguish AUGUST from APRIL). You can use this form for specifying the month only when using the "dd month year" format.

year Designation for the year. You can enter this as a two- or four-digit number, as follows:

entered year	actual year
0 through 77	2000 through 2077
78 through 99	1978 through 1999
100 through 1977	error
1978 through 2099	1978 through 2099
2100 and up	error

If you omit the date 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.

mm/dd/year Alternate form of the date parameter. If you use this form, you do not have to surround the parameter with quotes. The individual fields of this parameter are:

mm Numerical designation for the month (for example: 1 represents January, 2 represents February, etc.). You can use this form for specifying the month only when using the "mm/dd/year" format.

dd Same as in the previous form of the date parameter.

year Same as in the previous form of the date parameter.

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:

Entry	Action
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.

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, including owner
- Extension data
- 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 use 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 to 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 its 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.y
```

where Vx.y is the version number of the utility. It then displays the following message:

```
all files modified after <date>, <time> will be saved
```

where <date> and <time> are the values you specified in the date and time parameters (or the defaults). Then BACKUP prompts you for a backup volume.

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:

Entry	Action
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>, <time>, enter Y to overwrite:

or

<backup device>, named volume, <volume name>, enter Y to continue:

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

Entry	Action
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.

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

<pathname>, saved

or

<pathname>, directory saved

When the backup process is complete, BACKUP displays the number of data files saved, as follows:

```
files saved = <num>
```

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 restore files to a named volume with the RESTORE utility, you must supply the backup volumes in order.

ERROR MESSAGES

- <backup device>, backup operation not completed

  When BACKUP requested a new backup volume, you specified an "E" to exit BACKUP. This message is a reminder that the backup operation is not complete. The last file on the last backup volume may be incomplete.

- <backup device>, backup volume #<nn>, <date>, <time>, 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 value> : <exception mnemonic>

  <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 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 value> : <exception mnemonic>, 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>, device in use
  <backup device>, <exception value> : <exception mnemonic>

  The device you specified for the backup device is the same device that contains your input pathname. Continuing would result in damage to the files on the input volume.

- <backup device>, error writing volume label
  <backup device>, <exception value> : <exception mnemonic>

- <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 write-protected. The second line of the message indicates the iRMX 86 exception code encountered. BACKUP reprompts for a different backup volume.

- <backup device>, input and output are on same device

  The device you specified for the backup device is the same device that contains your input pathname. Continuing would result in damage to the files on the input volume.

- <backup device>, invalid backup device

  The logical name you specified for the backup device was not a logical name for a device. Examples of invalid names are :CI:, :CO:, and :HOME:.

- <exception value> : <exception mnemonic>, 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>, named volume, <volume name>, enter Y to overwrite:

  The backup volume you supplied is a named volume. BACKUP lists the logical name of the device containing the volume and the volume name. It overwrites this volume if you enter Y, y, R, or r.

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

  The backup volume you supplied is a formatted volume, but it has a label that is not readable. 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.

- <exception value> : <exception mnemonic>, requested date/time later than system date/time

  The date and time you specified is more recent than the current system date and time (as set by the DATE and TIME commands). Either the date and time you specified in the BACKUP command are in error or you did not set the system date and time.

- <pathname>, too many input pathnames

  You attempted to enter a list of pathnames or use a wild-carded pathname as the input pathname. You can enter only one pathname per invocation of BACKUP.

- <pathname>, too many output pathnames

  You attempted to enter a list of logical names for the backup device. You can enter only one output logical name per invocation of BACKUP.

- <pathname>, unable to complete directory

  BACKUP encountered an error when accessing a file in the <pathname> directory. It skips the rest of the files in the directory and goes on to the next directory. This error could occur if you do not have list access to the directory.

- <backup device>, volume not formatted

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

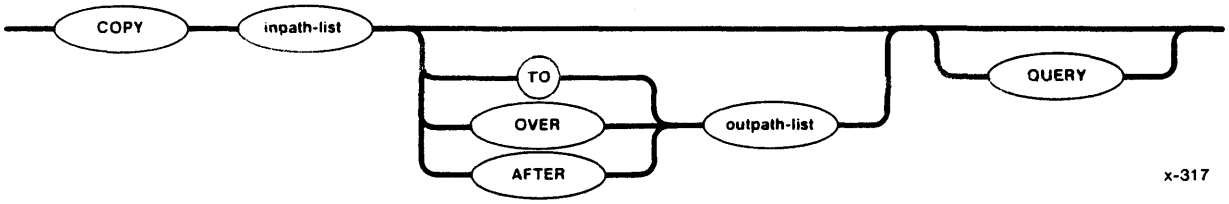
  The backup volume you supplied was not formatted. BACKUP continues to issue this message until you supply a formatted backup volume.

- <backup device>, write error on backup volume
  <backup device>, <exception value> : <exception mnemonic>

  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.

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 on 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:

Entry	Action
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 displays the following message: <pathname>, already exists, OVERWRITE? Enter Y, y, R, or r if you wish to write over the existing file. Enter an "N" (upper or lower case) or a carriage return alone if you do not wish to overwrite the existing file. In the latter case, the COPY command will pass over the corresponding input file without copying it, and will attempt to copy the next input file to its corresponding output file. If you specify multiple input files and a single output file, COPY appends the remaining input files to the end of the output file.
OVER	Writes the 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 corresponding output file name. If you specify multiple input files and a single output file, COPY appends the remaining input files to the end of the 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 you omit the preposition and outpath-list parameters, COPY displays the output at your console screen (TO :CO:).

DESCRIPTION

The COPY command can be used to perform several different operations. Some of these include:

- Creating new files (TO preposition).
- Copying over existing files or creating new files (OVER preposition).
- Adding data to the end of existing files (AFTER preposition).
- Copying a list of files to another list of files on a one-for-one basis.
- Concatenating two or more files into a single output file.

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

<pathname>, copied TO <out-pathname>

<pathname>, copied OVER <out-pathname>

<pathname>, copied AFTER <out-pathname>

When you copy files, the number of input pathnames you specify must equal the number of output pathnames, unless you specify only one output pathname. In the latter case, COPY appends the remainder of the input files to the end of the ouput file. As each file is appended, the following message is displayed on the console screen:

<pathname>, copied AFTER <output-file>

If you specify multiple output files, and there are more input files than output files, or if you specify fewer input files than output files, COPY returns an error message.

Also, if you specify a wild card character in an output pathname, you must specify the same wild card character in the corresponding input pathname. Other combinations result in error conditions.

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 :F1:/ALPHA/BETA
```

This would copy the A data file to a different volume, directory, and filename, where the new file's pathname would be :F1:/ALPHA/BETA.

The user ID of the user who invokes the COPY command is considered the owner of new files created by COPY. Only the owner can change the access rights associated with the file (refer to the PERMIT command later in this chapter).

When COPY creates new files, it sets the access rights and list of accessors as follows:

- It sets the file for ALL access (delete, read, append, and change).
- It sets the owner as the only accessor to the file.

Refer to the PERMIT command for more information about access rights and the list of accessors.

ERROR MESSAGES

- <pathname>, output file same as input file

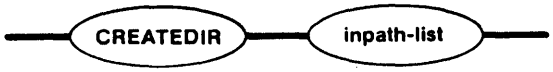
  You attempted to copy a file to itself.

- <pathname>, UPDATE or ADD access required

  Either you cannot overwrite the information in a file because you do not have update access to it, or you cannot copy information to a new file because you do not have add entry access to the file's parent directory.

CREATEDIR

This command creates one or more iRMX 86 user directories. The format is as follows:



x-318

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

CREATEDIR creates a directory with all access rights available to you, the owner. That is, you can delete, list, add, and change the contents of the directory you created with CREATEDIR. Other users (except the system manager) have no access to the directory unless you use the PERMIT command (described later in this chapter) to change the access rights and list of accessors.

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

causes the newly-created directory GH to be nested within existing directory EF, which in turn, is nested within directory DC, and so on. The directories AB, DC, and EF must already exist before entering this command.

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).

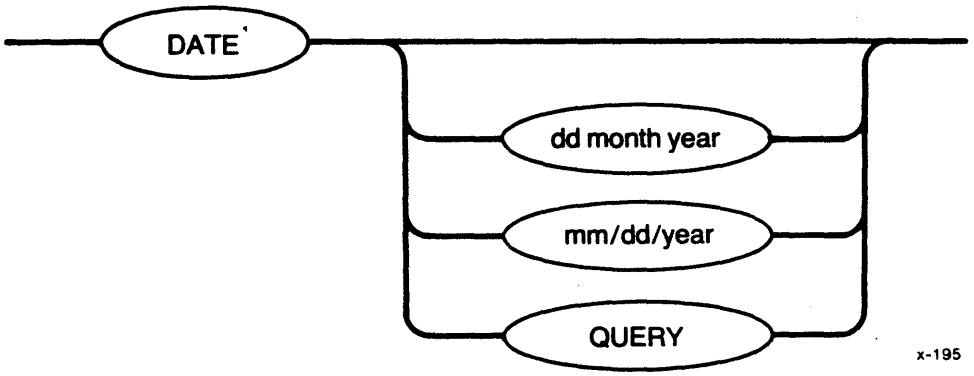
ERROR MESSAGE

- <directory-name>, file already exists

  The pathname of the directory to be created already exists.

DATE

This command sets a new system date or displays the current date and time. The format is as follows:



INPUT PARAMETERS

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

month — Designation for the month. You can enter the whol name (such as AUGUST) or enough characters to distinguish one month from another (for example, AU, to distinguish AUGUST from APRIL). You can us this form for specifying the month only when using the "dd month year" format.

mm — Numerical designation for the month (for example: represents January, 2 represents February, etc.). You can use this form for specifying the month onl when using the "mm/dd/year" format.

year — Designation for the year. You can enter this as a two- or four-digit number, as follows:

entered year	actual year
0 through 77	2000 through 2077
78 through 99	1978 through 1999
100 through 1977	error
1978 through 2099	1978 through 2099
2100 and up	error

QUERY — Causes DATE to prompt for the date by issuing the following message:

DATE:

DATE continues to issue this prompt until you ent a valid date.

DESCRIPTION

If you set one date parameter, you must set all three; there are no default settings for individual date parameters. You must separate the dd, month, and year entries with single blanks.

If you omit the date parameters, DATE displays the current date and time in the following form:

```
dd mmm yy, hh:mm:ss
```

When the Operating System displays the date, it displays only the first three characters of the month and the last two digits of the year. It separates the hours, minutes, and seconds of the time with colons.

If you request the date on a non-timing system, DATE displays the following message:

```
00:00:00
```

Refer to the TIME command in this chapter if you wish to set the system clock while setting the date.

ERROR MESSAGES

- <date>, invalid date

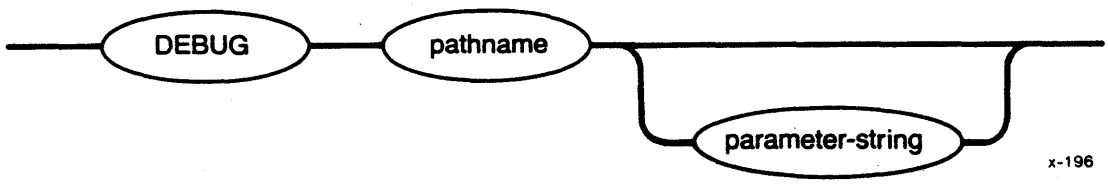
  You entered an invalid date. This could result from specifying a day that is invalid for the month you specified (such as 31 FEB 82), entering characters for the year parameter that do not fall into the legitimate ranges listed under the year parameter, entering a month parameter that does not uniquely identify the month, or entering invalid characters.

- <parameter>, invalid syntax

  You specified both a date and the QUERY parameter in the DATE command.

DEBUG

This command allows you to debug your iRMX 86 application jobs in conjunction with the iSBC 957B hardware package and monitor.



INPUT PARAMETERS

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 957B monitor. You can then use the iSBC 957B monitor to single-step, display registers, and set breakpoints within the program. Refer to Appendix C for a summary of monitor commands, and to the USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE for a complete description of the iSBC 957B functions.

When you invoke the DEBUG command, it displays the following message:

```
DEBUG file, <pathname>
```

where <pathname> is the pathname of the file containing the application job to debug. Then DEBUG loads the application job and displays information about the location of the job's segments and groups. Figure 3-1 shows an example of this output.

The first line of the display lists the token for the application job. The remaining lines list the base portions of all segments and groups created by LINK86 when the code was linked. The S(n) and G(n) values are the same as those that appear on the link map. Therefore, you can match the base values shown in this display with the offset values shown in the link map to determine the exact location of a symbol listed in the link map. Refer to the iAPX 86, 88 FAMILY UTILITIES USER'S GUIDE for information about LINK86 and the link map.

```
SEGMENT AND GROUP MAP FOR JOB:  A88F

NAME  BASE    NAME  BASE    NAME  BASE    NAME  BASE    NAME  BASE
S(1)  9E4E    S(2)  9E32    S(3)  9CFF    S(5)  9CEC    S(6)  A863
S(7)  A229    S(8)  A84D    S(9)  A152    S(13) 9C91    S(15) 9C85
S(17) 9C67    S(18) 9C5C

G(1)  A229    G(2)  A152
```

Figure 3-1. Sample DEBUG Display

When DEBUG executes, the iSBC 957B package disables interrupts. This causes the time-keeping function to stop when code is not executing. This slowing of the timing function:

- Affects the ability of the Operating System to keep track of the time-of-day and write its data structures to secondary storage.
- Stops type-ahead from working (see LINE EDITING in Chapter 2).
- Affects the ability of the system to execute time-out tasks that have provided time limits to system calls (this applies to users that are interfacing to the Operating System with system calls other than the UDI system calls.

Unless you use the monitor's NQ command to single-step through code, the iSBC 957B package cannot tolerate interrupts while single-stepping. The NQ command disables interrupts while single-stepping, allowing you to single-step through code without being interrupted by the system clock.

When DEBUG is invoked to debug an application program, it loads the application program into its own dynamic memory. This means that the application program obtains dynamic memory from the memory pool of DEBUG, not from the memory pool of the user session. Therefore, programs that experience problems with insufficient memory when run independently might not experience those problems when run under the control of DEBUG.

ERROR MESSAGE

- <exception value> : <exception mnemonic>, command aborted by EH

  While processing, the DEBUG command encountered an exceptional condition. Therefore, the Human Interface's exception handler aborted the command. The message lists the exception code that occurred.

DELETE

This command removes data files and empty directories from secondary storage. The format is as follows:

x-319

INPUT PARAMETERS

inpath-list — One or more pathnames for the named data 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:

Entry	Action
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. To delete a file, you need not be the owner of the file; however you must have delete access to the file. If a command or other program is accessing the file (has a connection to the file) when you enter the DELETE command, DELETE marks the file for deletion and deletes it when all connections to the file are gone.

Non-empty directories cannot be deleted. If you wish to delete a directory that contains files, you must first delete all its contents. For example, if you wish to delete a directory named ALPHA whose entire contents consist of a directory BETA containing a data file SAMP, you would enter the following command:

```
DELETE ALPHA/BETA/SAMP, ALPHA/BETA, ALPHA
```

This would delete all the files contained under ALPHA before deleting the directory itself.

DELETE displays the following message as it deletes each file or marks the file for deletion:

```
<pathname>, DELETED
```

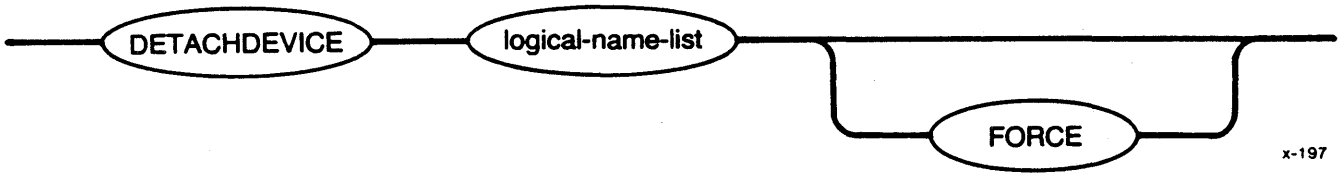
ERROR MESSAGE

- <pathname>, DELETE access required

  You do not have permission to delete the specified file.

DETACHDEVICE

This command detaches the specified devices and deletes their logical names. The format of this command is as follows:



INPUT PARAMETER

logical-name-list	One or more logical names of the physical devices that are to be detached. Colons surrounding each logical name are optional; however, if you use colons, you must use matching colons. Multiple logical names must be separated by commas.
FORCE	Causes DETACHDEVICE to detach the device even if connections to files on the device currently exist.

DESCRIPTION

The DETACHDEVICE command allows you to detach a device. After a device is detached, no volume mounted on that device is accessible for system use.

Unless you are the system manager (user ID 0), you can detach only the following devices:

- Devices that were attached with your user ID or WORLD (65535) as the owner ID
- Devices you originally attached using the ATTACHDEVICE command
- Devices originally attached using the WORLD parameter of ATTACHDEVICE

DETACHDEVICE returns an error message if you attempt to detach devices originally attached by other users. This prevents users from detaching devices belonging to other users and from accidentally detaching system volumes. However, the system manager can detach all devices.

Unless you specify the FORCE parameter, you cannot detach a device if any connections exist to files on the device (that is, if other users are currently accessing the device). However, the FORCE parameter causes DETACHDEVICE to delete all connections to files on the device before detaching the device.

After detaching the device and deleting its logical name, the DETACHDEVICE command displays the following message:

<logical-name>, detached

NOTE

Using the DETACHDEVICE command to detach the device containing your Human Interface commands prevents using Human Interface functions until the system is reinitialized.

ERROR MESSAGES

- <logical name>, can't detach device
  <logical name>, <exception value> : <exception mnemonic>

  An exceptional condition occurred which prevented DETACHDEVICE from detaching the device. This message lists the resulting exception code.

- <logical name>, device does not belong to you

  The device was originally attached by a user other than WORLD or you. Thus you cannot detach the device.

- <logical name>, device has outstanding file connections

  There are existing connections to files on the device. Because you did not specify the FORCE parameter, DETACHDEVICE does not detach the device.

- <logical name>, device is in use

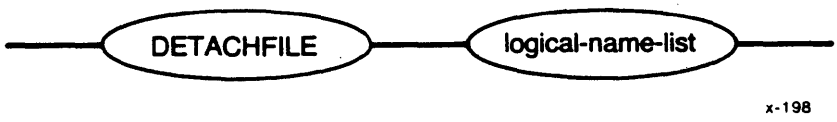
  Another user or program is accessing the device (has a connection to a file). Therefore, you must specify the FORCE parameter in order to detach the device.

- <logical name>, outstanding connections to device have been deleted

  There were outstanding connections to files on the volume. However, because you specified the FORCE parameter, DETACHDEVICE deleted those connections. This is a warning message that does not prevent DETACHDEVICE from detaching the device.

DETACHFILE

This command allows you to terminate the association of a logical name with a file. The format of this command is as follows:



INPUT PARAMETER

logical-name-list	List of logical names, separated by commas, that represent the files to be detached. Each logical name must be contain 1 to 12 characters. Colons surrounding each logical name are optional; however, if you use colons, you must use matching colons.

DESCRIPTION

You establish an association between a file and a logical name by entering the ATTACHFILE command. DETACHFILE breaks this association. It does this by deleting the logical name. When DETACHFILE detaches a file in this manner, it displays the following message:

<logical name>, detached

where <logical name> is the name you specified.

You cannot use DETACHFILE to detach devices. DETACHFILE returns an error message if you make such an attempt.

You cannot use DETACHFILE to detach logical names originally created by other users. DETACHFILE searches for logical names associated with your interactive job only.

ERROR MESSAGES

- <exception value> : <exception mnemonic> invalid global job

  The Human Interface encountered an internal system problem when it attempted to remove the logical name from the global job's object directory. The message lists the resulting exception code.

- <logical name>, logical name does not exist

  The logical name is not part of your interactive job.

- <logical name>, logical name not allowed

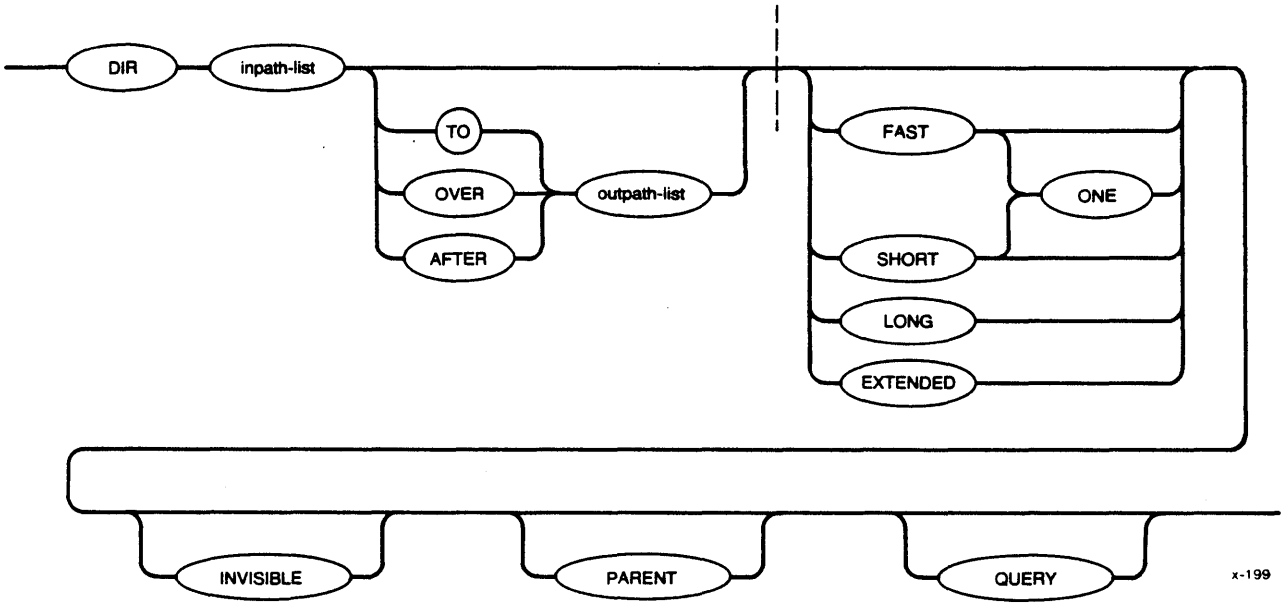
  The logical name you specified was either :$:, :HOME:, :CI:, or :CO:. You cannot detach the files associated with these logical names.

- <logical name>, not a file connection

  The logical name you specified is not the logical name of a file.

DIR

This command lists the names and attributes of the data and directory files contained in a given directory. The format of the command is as follows:



INPUT PARAMETERS

inpath-list	One or more pathnames of the directories to be listed (the pathnames can represent data files if the PARENT parameter is also specified). 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.
FAST	Lists only the filenames and directory names in the directory. The output format contains five columns of filenames unless you also specify the ONE parameter (see Figure 3-2 at the end of this command description). FAST is the default if you omit the listing format.
SHORT	Lists the file information in a two-column format (see Figure 3-3 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.

LONG — Lists file information in a one-line format (see Figure 3-4 at the end of this command description).

EXTENDED — Lists all available information for each data file or directory file in the directory. The first line for each file is the same as for the LONG form. The second line contains the last access date, creation date, and the accessor list. The listing is in a double-column format (see Figure 3-5 at the end of this command description).

INVISIBLE — Lists the invisible files (those beginning with the characters "R?" or "r?") in addition to the rest of the files in the directory. If you omit this parameter, DIR does not display invisible files.

PARENT — Causes DIR to display an entry for the directory specified in the inpath-list in addition to the files contained in the directory. This parameter is useful for obtaining information about the root directory of a volume when using the LONG or EXTENDED parameters.

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:

Entry	Action
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 the destination file already exists, DIR displays the following information:

```
<pathname>, already exists, OVERWRITE?
```

Enter Y, y, R, or r if you wish to delete the existing file. Enter any other character if you do not wish to delete the file.

If you omit the TO/OVER/AFTER preposition and the output pathname, 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.

outpath-list — One or more pathnames of the files to receive the directory listing. Multiple pathname entries must be separated by commas. Separating blanks are optional. If you omit the preposition and the outpath-list, the default destination is the user's console screen (TO :CO:).

DESCRIPTION

You do not need to be the owner of a directory to list its contents with DIR; however, you must have LIST access to the directory.

The amount of information listed for each file depends upon what listing format you specify (FAST, SHORT, LONG, or EXTENDED) in the DIR command. An example of each type of listing format is provided at the end of the DIR command description in Figures 3-2 through 3-5 respectively. Table 3-3, which follows the figures, provides an explanation of the illustrated headings.

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
```

displays a listing of the default directory in the EXTENDED format. Note that your default directory is determined by your user ID.

Figures 3-2, 3-3, 3-4, and 3-5 show output examples for FAST, SHORT, LONG, and EXTENDED listing formats respectively. Table 3-3 defines the displayed column headings.

If a file name begins with the characters "R?" or "r?", it is an invisible file. Normally DIR does not display invisible files. However, you can specify the INVISIBLE parameter to display these files.

If you do not know the name of the directory that contains a file (the file's parent directory), you can still display its contents by using the PARENT parameter. Rather, it displays the parent directory of the file you specify.

If you use the TO preposition to copy the output of the DIR command to a file and specify the pathname of an existing file, DIR displays the following information:

    <pathname>, already exists, OVERWRITE?

Enter Y, y, R, or r if you wish to delete the existing file. Enter any other character if you do not wish to delete the file.

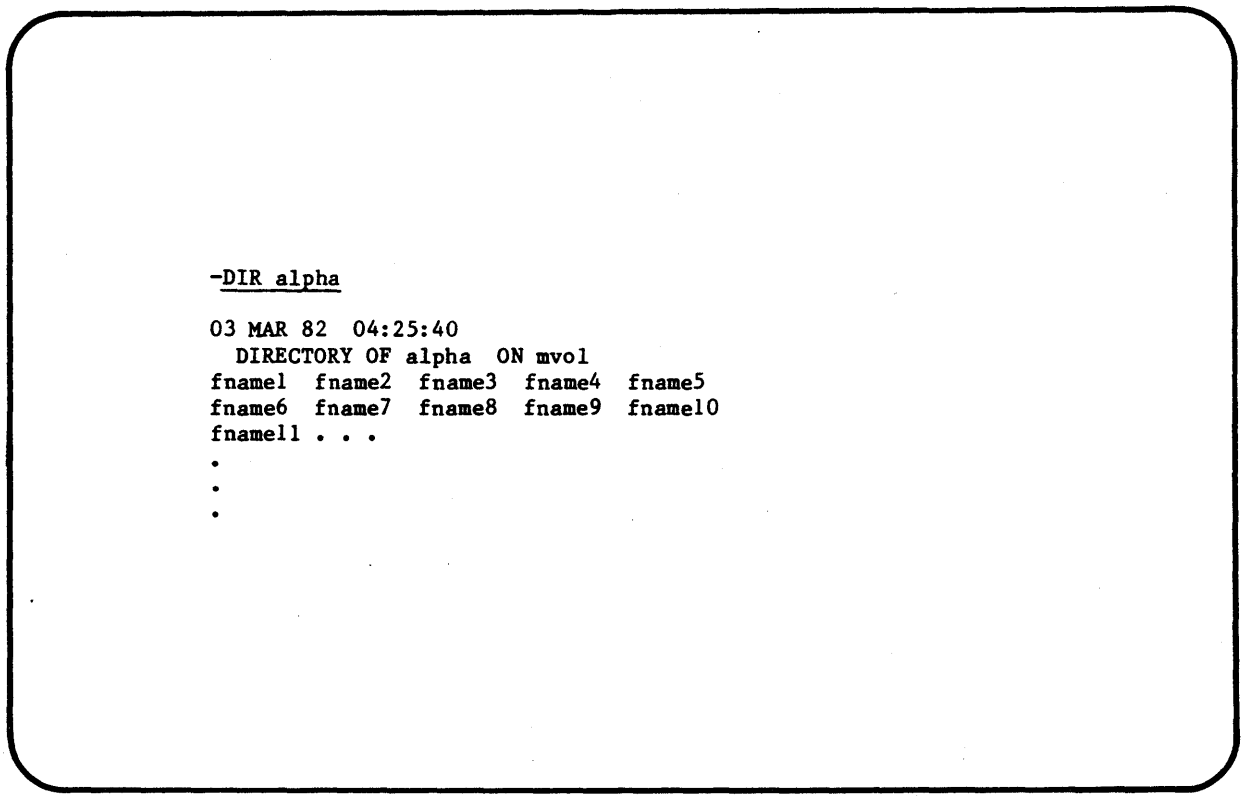


Figure 3-2. FAST Directory Listing Example (Default Listing Format)

```
-DIR mydirectory2 S

03 MAR 82   21:55:24
DIRECTORY OF mydirectory2 ON myvol
  NAME          AT   ACC  BLKS  LENGTH     NAME           AT   ACC  BLKS  LENGTH
append              -R--     2  1425       alpha.obj           DRAU     3   2871
REFERENCE       DR  -L--     1    10       DATA           DR   DLAC     1      4
LEMONADEIT          DRAU  123456789 123456789
time                DRAU     6  5374       detachdevice        DRAU     4   3414
test                -R--     5  4415       schedule            ---U     7   6976
testprog.a86        -RA-     2  2040       DATABASE.LST        -RAU    11  10336
EXPERIMENTAL    DR  -LAC     1    20       BACKUP         DR   DLAC     1     10

     13 FILES     44 BLOCKS        36895 BYTES
```

Figure 3-3. SHORT Directory Listing Example

```
-DIR mydirectory1 L

03 MAR 82   21:55:24
DIRECTORY OF mydirectory1 ON myvol

                                                    GRAN
   NAME         AT   ACC    BLKS       LENGTH     VOL FIL OWNER          LAST MOD
ed                  -R--      11         1057    1024   1 # 47         02 MAR 82
programs        DR  DL--      30        30185    1024   1 # 47         03 MAR 82
fmat                DRAU       1           39    1024   1 # 655535     08 NOV 81
OBJFILE             ---U       3         2895    1024   1 # 47         18 DEC 81
ALPHA1.P86          DLAC       2         1304    1024   1 # 50         22 OCT 81
ALPHA1.MP1          DLAC       6         5397    1024   1 # 50         22 OCT 81
manuals         DR  -L--       1          304    1024   1 # 47         02 JUL 80

      7 FILES     54 BLOCKS     41181 BYTES
```

Figure 3-4. LONG Directory Listing Example

```
-DIR mydir E

03 MAR 82   21:55:24
DIRECTORY OF mydir ON myvol
                                                    GRAN
   NAME         AT   ACC    BLKS      LENGTH     VOL FIL OWNER            LAST MOD
programs        DR   DL--     30       30185    1024   1 # 47             03 MAR 82
                          CREATION: 01 JAN 81  04:05:44      ACCESSORS         ACC
                          LAST ACC: 03 MAR 82  05:52:33      # 47              DL--
                          LAST MOD: 03 MAR 82  05:52:33      # 50              -LA-
                                                             # 82              -L--
ed                   -R--     11        1057    1024   1 # 47             02 MAR 82
                          CREATION: 11 NOV 81  12:24:05      ACCESSORS         ACC
                          LAST ACC: 02 MAR 82  14:22:16      # 47              -R--
                          LAST MOD: 02 MAR 82  14:22:16
fmat                 DRAU      1          39    1024   1 # 65535          08 NOV 81
                          CREATION: 01 NOV 81  08:54:39      ACCESSORS         ACC
                          LAST ACC: 03 MAR 82  14:56:59      # 65535           DRAU
                          LAST MOD: 08 NOV 81  20:44:01
testdir         DR   DLAC     1          32    1024   1 # 47             01 MAR 82
                          CREATION: 02 FEB 82  15:02:42      ACCESSORS         ACC
                          LAST ACC: 03 MAR 82  09:32:53      # 47              DLAC
                          LAST MOD: 01 MAR 82  13:13:07      # 50              -LA-
                                                             # 65535          -L--

        4 FILES      43 BLOCKS      32213 BYTES
```

Figure 3-5. EXTENDED Directory Listing Example

Table 3-3. Directory Listing Headings

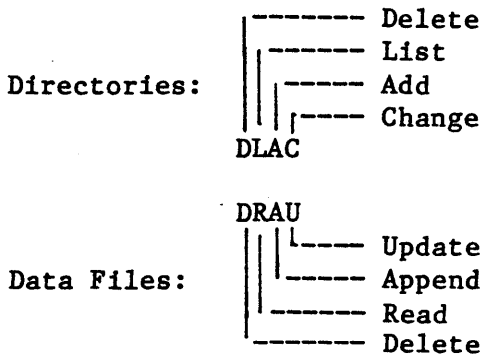
Heading	Meaning
NAME	14-character field for the file name.
AT	File attribute, where: DR = Directory MP = Bit map file blank = Data file
ACC	File access rights of the user who entered the DIR command, where: 
BLKS	Up to nine-digit number (five digits on SHORT listing) giving the volume-granularity units allocated to the file. On the SHORT display, if the number of digits exceeds five, DIR displays the file in the nine-digit form (see the LEMONADEIT file in Figure 3-3).
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 LEMONADEIT file in Figure 3-3).
VOL	Five-digit number giving the volume granularity in bytes.
FIL	Three-digit number giving the granularity of the file in multiples of volume granularity.
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.

Table 3-3. Directory Listing Headings (continued)

Heading	Meaning
ACCESSORS	User IDs of users who have access to the file.
ACC	Access rights of the corresponding user. The format of this field is identical to ACC as described previously.

ERROR MESSAGES

- no directory files found

  None of the files you specified were directories.

- <pathname>, READ access required

  You do not have read (list) access to the directory.

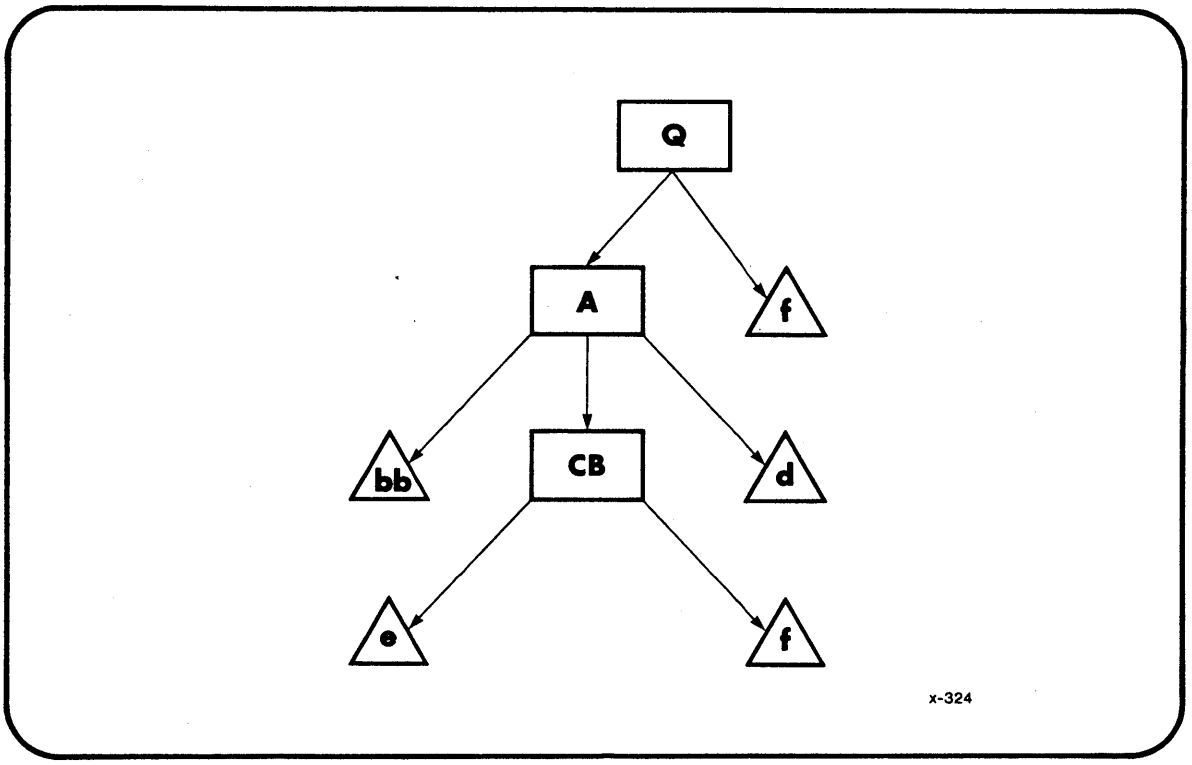
- <pathname>, UPDATE or ADD access required

  Either you cannot overwrite the information in a file because you do not have update access to it, or you cannot copy information to a new file because you do not have add entry access to the file's parent directory.

EXAMPLES

The examples that follow show how a directory's files are listed when you use your 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:



Example 1:

Suppose your default prefix is :F0:Q. This example shows the files that would be listed in response to various DIR commands. It shows the pathnames that you could enter and the resulting files that DIR would list.

Pathname	Files Listed
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

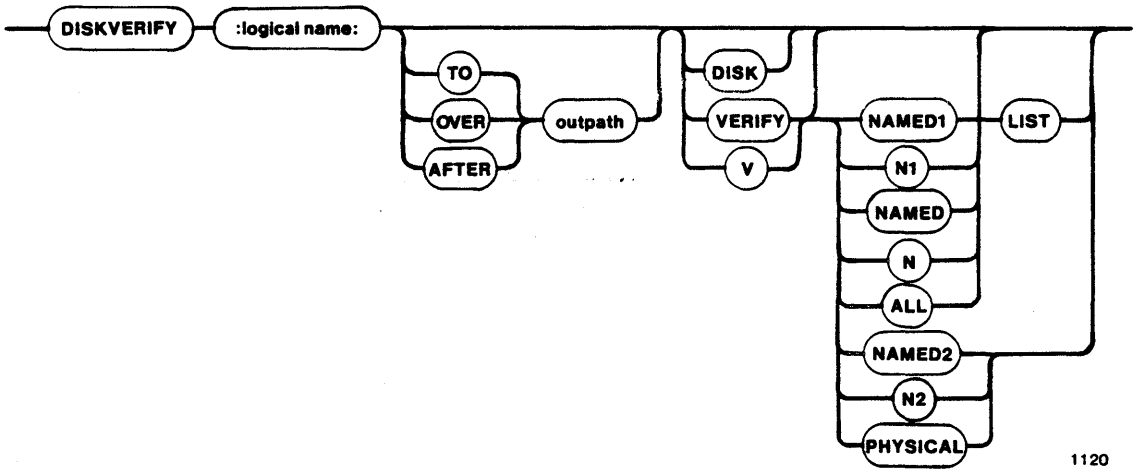
Example 2:

Suppose your default prefix is :F0:Q/A. This example also shows the files that would be listed in response to various DIR commands.

Pathname	Files Listed
omitted	bb, CB, d
A	not allowed because directory A does not contain an entry A
CB	e, f

DISKVERIFY

This command invokes a 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.

DISK Displays the attributes of the volume (such as type of volume, device granularity, block size, number of blocks, interleave factor, extension size, volume size, and number of fnodes) and returns control to you at the Human Interface level. You can then enter any Human Interface command.

If you omit this parameter (and the VERIFY 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.

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 (and the DISK 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.
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 hierarchy. (Refer to the description of the FORMAT command for more information about fnodes.) 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.
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.
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 only the PHYSICAL verification function.
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.
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.

LIST	VERIFY option that you can use with other VERIFY options that, either explicitly or implicitly, specify the NAMED1 option. When you use this option, the file information generated by VERIFY is displayed for every file on the volume, even if the file contains no errors. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.

OUTPUT PARAMETERS

TO	Copies the output from the disk verification utility to the specified file. If the file already exists, DISKVERIFY displays the following information: <pathname>, already exists, OVERWRITE? Enter Y, y, R, or r to write over the existing file. Enter any other character if you do not wish to overwrite the 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.
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 error message.

DESCRIPTION

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

iRMX 86 DISK VERIFY UTILITY, Vx.y

where Vx.y is the version number of the utility. If you specify the VERIFY or DISK parameter in the DISKVERIFY command, the utility performs the operation specified in the parameter and copies the output 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 output. After generating the output, the utility returns control to the Human Interface, which prompts you for more Human Interface commands. The following is an example of a DISKVERIFY command that uses the VERIFY option:

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

'NAMED2'  VERIFICATION
    BIT MAPS O.K.
-
```

The following is an example of a DISKVERIFY command that uses the DISK option:

```
-DISKVERIFY :F2: DISK
iRMX 86 DISK VERIFY UTILITY, Vx.y
Device name = WF0
   Named disk, Volume name = UTILS
      Device gran = 0080
       Block size = 0080
     No of blocks = 0000072D : No of Free blocks = 00000408
      Volume size = 0003E900
       Interleave = 0005
   Extension size = 03
     No of fnodes = 0038       : No of Free fnodes = 0022
-
```

However, if you omit the VERIFY and DISK parameters 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

- argument error

  The VERIFY option you specified is not valid.

- command syntax 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.

- not a named disk

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

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.

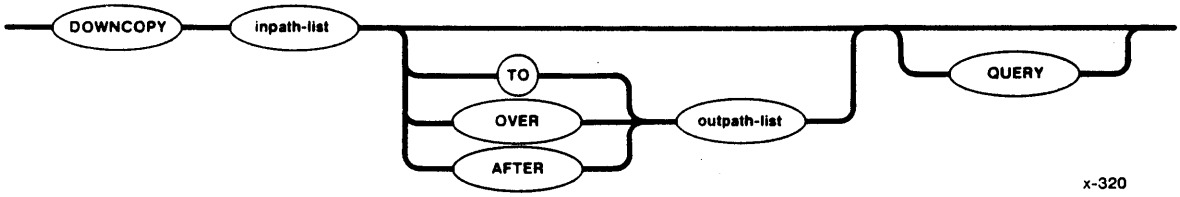
EXAMPLE

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

```
-DISKVERIFY :F1: VERIFY ALL

iRMX 86 DISK VERIFY UTILITY, Vx.y
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 957B 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 <outfile>?

<pathname>, copy down OVER <outfile>?

<pathname>, copy down AFTER <outfile>?

Enter one of the following in response to the query:

Entry	Action
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.

OUTPUT PARAMETERS

TO	Reads iRMX 86 files and copies them TO new ISIS-II files in the listed sequence. If the specified output files already exist in the ISIS-II directory when the TO parameter is used, DOWNCOPY displays the following message: <filename>, already exists, OVERWRITE? Enter Y, y, R, or r 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 are appended to the end of the last-specified ISIS-II file.
OVER	Copies the iRMX 86 input files OVER the existing ISIS-II destination files in the specified sequence. If you specify multiple input files and one output file, DOWNCOPY appends the remaining input files to the end of the output file.
AFTER	Copies the iRMX 86 input files, in sequence, AFTER the end of data on the existing ISIS-II destination files.
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 goes to the ISIS-II console screen.

DESCRIPTION

The DOWNCOPY command cannot be used to copy directories from an iRMX 86 system to a Series III microcomputer development system; only files can be copied.

Before you enter a DOWNCOPY command on the iRMX 86 console keyboard, your target system must be connected to a Series III system via the iSBC 957B package, and the package must be running. To do this, you must start your iRMX 86 system from the Series III terminal (either by loading the software into the target system and using the monitor G command to start execution, or by using the monitor B command to bootstrap load the software). DOWNCOPY does not function if you start up your system from the iRMX 86 terminal or if you establish the link between the Series III system and target system after starting up your iRMX 86 system.

When DOWNCOPY copies files to the development system, it turns off all ISIS-II file attributes.

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:):

<pathname>, copied down TO <out-filename>

<pathname>, copied down OVER <out-filename>

<pathname>, copied down AFTER <out-filename>

When the DOWNCOPY command is executing, the iSBC 957B package disables interrupts. This affects services such as the time-of-day clock. Also, the Operating System is unable to receive any characters that you type-ahead while the DOWNCOPY command is executing.

ERROR MESSAGES

- <pathname>, DELETE access required

  DOWNCOPY could not replace an existing ISIS-II file because the file is write-protected.

- <pathname>, ISIS ERROR: <nnn>

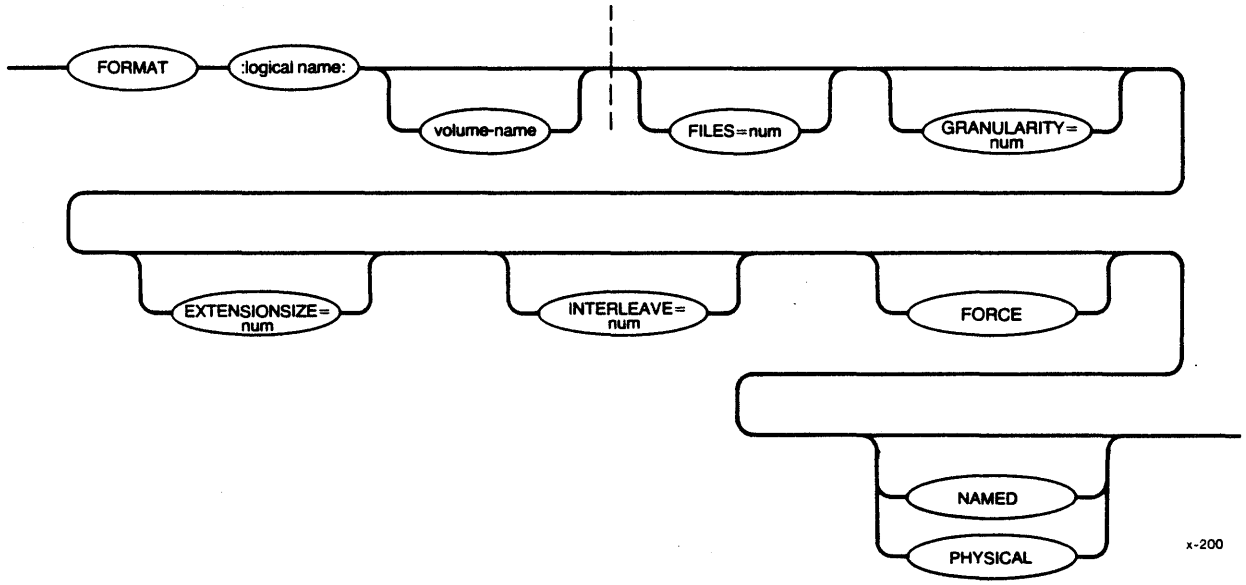
  An ISIS-II Operating System error occurred when DOWNCOPY tried to transfer the file to the Microcomputer Development System. Refer to the INTELLEC SERIES III MICROCOMPUTER DEVELOPMENT SYSTEM CONSOLE OPERATING INSTRUCTIONS for a description of the resulting error code.

- ISIS link not present

  The iRMX 86 system is not connected to the development system via the iSBC 957B package.

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. You must surround the logical name with colons. Also, you must not leave space between the logical name and the succeeding volume name parameter.
volume-name	Six-character, alphanumeric ASCII name, without embedded blanks, to be assigned to the volume. If you include this parameter, you must not leave spaces between the logical name and the volume name.
FILES=num	Defines the maximum decimal number of user files that can be created on a NAMED volume. (This parameter is not meaningful when formatting a PHYSICAL volume and is ignored if specified for such volumes.) FORMAT uses the information specified in this parameter to allocate space for the number of files that will be created on the NAMED volume. The range for the FILES parameter is 1 through 32,761, although the maximum number of user files you can define depends on the settings of the GRANULARITY and EXTENSIONSIZE parameters (as explained in the "Description" portion of this command write-up). When you use this parameter, FORMAT creates room for six additional files for internal system use. If not specified, the default is 50 user files.

FORCE — Forcibly deletes any existing connections to files on the volume before formatting the volume. If you do not specify FORCE, you cannot format the volume if any connections to files on the volume still exist.

GRANULARITY=num — Volume granularity; the minimum number of bytes to be allocated each time a file's size is increased on a NAMED volume. (This parameter is not meaningful for PHYSICAL volumes, and is ignored if specified for such volumes.) FORMAT rounds the value you specify up to the next multiple of the device granularity. Then it places the decimal number in the header of the volume, where it becomes the default file granularity when a file is created on the volume. The range is 1 through 65,535 (decimal) bytes, although the maximum allowable volume granularity depends on the settings of the FILES 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.

NOTE

Using a large volume granularity (in excess of 1024), might cause users to exceed their memory limits when executing programs that reside on the volume. This can occur because the Operating System uses the volume granularity as a minimum buffer size when reading and writing files.

EXTENSIONSIZE=num — Size, in bytes, of the extension data portion of each file. (This parameter is not meaningful for PHYSICAL volumes, and is ignored if specified for such volumes.) The range is 0 through 255 (decimal), although the maximum allowable extension size depends on the settings of the FILES 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. Acceptable values are 1 through 255 decimal. If not specified, the default value is 5. 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 file 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 FILES parameters are not meaningful when PHYSICAL is specified for the volume. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the file type 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 display the root directory of the volume, and the name appears in the directory's heading. Although the Human Interface uses the volume name in its own internal processing when you access the volume, you need not specify the volume name in any subsequent command after the volume is formatted. You must specify only the logical name of the secondary storage device that contains the volume.

Volume Name

The volume name allows you to identify a mass storage volume by a recorded name. You will see this name when you ask for a DIRectory listing of any directory on the volume. For diskettes, a volume name gives you a method for identifying a volume in case the stick-on label on the diskette gets lost or destroyed.

Once the volume is formatted, you do not need to specify the volume name in commands -- you identify the volume with its logical name.

Files

You can specify the number of files reserved for user files with the FILES parameter. Each time you create a file on the volume, the Operating System records information about the file in an unused area of the volume, and later uses this information to determine the location of the file on the volume.

Internal Files

When you format a named volume, FORMAT creates six internal system files. It names three of these files and lists their names in the root directory of the volume. The files are:

file	description
R?SPACEMAP	Volume free space map
R?FNODEMAP	Free fnodes map
R?BADBLOCKMAP	Bad blocks map

It grants the user WORLD read access to these files. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about these files.

Root Directory

The root directory lists the user who formats the volume as the owner, giving that user all access rights. No other user has access to the root directory until the owner explicitly grants access. The owner can grant other users access to the volume via the PERMIT command described later in this chapter. However, because the owner has all access rights to the root directory, the owner can obtain exclusive access to the volume, and can obtain delete access to any file created on the volume, even files created by other users.

Extension Data

Each file 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 AGETEXTENSION$DATA and A$SET$EXTENSION$DATA system calls (refer to the iRMX 86 BASIC I/O SYSTEM 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 255 bytes, the Human Interface requires all fnodes to have at least 2 bytes of extension data.

Volume Granularity

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 permanent storage to each new file you create on that volume in multiples of 128 bytes, regardless of whether the file requires the full amount.

Relationship between FILES, GRANULARITY, and EXTENSIONSIZE

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

$$(87 + \text{EXTENSIONSIZE}) \times (\text{FILES} + 6) / \text{GRANULARITY} \leq 65535$$

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

Interleave Factor

The interleave factor applies to volumes formatted either for NAMED or PHYSICAL files. The interleave factor specifies the logical sector sequence. The interleave specification maximizes access speed for the files on a given volume by matching the time it takes to read sequential sectors to time it takes the system to process the data. For example, an interleave factor of 5 for a flexible disk drive means that, for each file, the I/O System reads and writes every fifth sector on the diskette, starting with an index of 1. (Hard disk systems may be different.) With the appropriate interleave factor, 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.

Output Display

The FORMAT command displays one of the following messages when volume formatting is completed. For physical volumes:

```
volume (<volume name>) will be formatted as a PHYSICAL volume
    device gran.        = <number>
    interleave          = <number>
    volume size         = <k-number> K (or M)
volume formatted
```

For named volumes:

```
volume (<volume name>) will be formatted as a NAMED volume
    granularity       = <number>          sides     = <sides>
    interleave        = <number>          density   = <density>
    files             = <number>          disk size = <d-size>
    extensionsize     = <number>
    volume size       = <k-number> K (or M)
volume formatted
```

where:

<volume name>	Volume name specified in the FORMAT command.
<number>	Decimal number as specified in the command (or the default)
<k-number>	Volume size in K (1024-byte units) or M (1048576-byte units). FORMAT displays the volume size in Kbyte units unless the size is greater than 25 Mbytes.
<sides>	Number of sides of the volume that will be formatted (1 or 2). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<density>	Density at which the volume will be formatted (single or double). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<d-size>	Size of the volume (8 or 5.25). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.

ERROR MESSAGES

- <logical name>, can't attach device
  <logical name>, <exception value> : <exception mnemonic>

  FORMAT cannot attach the device for formatting, or it cannot re-attach the device (that is, restore it to its original condition) after formatting takes place.

- <logical name>, can't detach device
  <logical name>, <exception value> : <exception mnemonic>

  FORMAT cannot detach the device for formatting, 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.

- <logical name>, device is in use

  You cannot format the volume because there are outstanding connections to files on the volume and you did not specify the FORCE parameter.

- <vol-name>, fnode file size exceeds 65535 volume blocks

  The values you specified for fnode size, granularity, and extension data size cause the formula listed in the "Description" section to exceed its limit.

- <number>, invalid number

  You specified an out-of-range number for any of the FILES, GRANULARITY, EXTENSIONSIZE, or INTERLEAVE parameters.

- <logical name>, outstanding connections to device have been deleted

  There were outstanding connections to files on the volume. However, because you specified the FORCE parameter, FORMAT deleted those connections. This is a warning message that does not prevent FORMAT from formatting the volume.

- 0085 : E$LIST, too many values

  You entered multiple logical-name/volume-name combinations separated by commas. FORMAT can format only one volume per invocation.

- <volume name>, volume name is too long

  FORMAT requires the volume name you specify to be 6 characters or less.

INITSTATUS

This command displays the initialization status of Human Interface terminals. The format of this command is as follows:

DESCRIPTION

INITSTATUS displays at the user terminal the initialization status of all Human Interface terminals. Figure 3-6 illustrates the format of the INITSTATUS display.

TERMINAL DEVICE NAME	CONFIG EXCEP	DEVICE EXCEP	INIT EXCEP	USER STATE	JOB ID	USER ID
.T0.	0000	0000	0000	LE	1	65535
.T1.	0000	0000	0000	-E	2	1
.T3.	0000	0002		--		
.T4.	0021			--		

Figure 3-6. INITSTATUS Display

The columns listed in Figure 3-6 contain the following information.

TERMINAL DEVICE NAME — The physical name of the terminal, as defined when the iRMX 86 PC System was configured. Periods surround each name.

CONFIG EXCEP — Hexadecimal condition code that the Human Interface received when it attempted to interpret the terminal definition and user definition files (refer to Chapter 6 for more information). A zero value indicates a normal condition. Nonzero values indicate exceptional conditions. Refer to Appendix B for a list of exception codes.

DEVICE EXCEP — Hexadecimal condition code that the Human Interface received when it originally attached the terminal as a physical device.

INIT EXCEP — Condition code that the Human Interface received when it created a job for the interactive session.

USER STATE — Two characters that indicate the current state of the terminal. The first character can be either:

- L — The terminal is locked and cannot be reinitialized (refer to the LOCK command later in this chapter).
- \- — The terminal is unlocked.

The second character can be either:

- E — The Human Interface created the interactive job associated with this terminal and the job exists.
- \- — The interactive job does not exist.

JOB ID — A sequential number that the Human Interface assigns to the interactive job during initialization. You must specify this number as a parameter in the JOBDELETE command in order to delete the corresponding interactive job.

USER ID — User ID associated with the interactive job. This is the identification of the user that the Human Interface associates with the job when the user begins a Human Interface session.

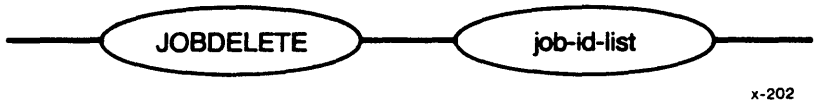
ERROR MESSAGE

- not a multi-access system

  The Human Interface cannot return information about terminals because it is not configured for multi-access. This message will not be returned for the Preconfigured iRMX 86 Operating System.

JOBDELETE

This command deletes a running interactive job. The system manager can use this command to delete any interactive job. Other users can delete only those interactive jobs that have the same user ID that they have. The format of this command is as follows:



where:

job-id-list	One or more job IDs, separated by commas, of the interactive jobs to be deleted. You can obtain the IDs of jobs by invoking the INITSTATUS command (described earlier in this chapter).

DESCRIPTION

The JOBDELETE command allows users to delete interactive jobs. Deleting an interactive job causes the Human Interface to terminate the corresponding user session.

When JOBDELETE attempts to delete a job, it first attempts to delete the job's offspring jobs (for example, a SUBMIT file). It deletes multiple levels of offspring jobs. However, JOBDELETE cannot delete any interactive job (or offspring) that contains extension objects. Refer to the iRMX 86 NUCLEUS REFERENCE MANUAL for information about extension objects.

Normally, when a user's interactive job is deleted, the Human Interface recreates the interactive job, thus restarting the user session. However, if the LOCK command (described later in this chapter) has been specified for the user's terminal, the Human Interface does not automatically recreate the user's interactive job after a JOBDELETE command. Therefore, the system manager can use the combination of LOCK and JOBDELETE to remove users from the system prior to a system shutdown.

As JOBDELETE deletes each job, it displays the following message at the user terminal (:CO:):

<job-ID>, deleted

where <job-ID> is the identifier of the deleted job.

ERROR MESSAGES

- <job-ID>, does not exist

  The interactive job associated with the identifier <job-ID> does not exist. It has already been deleted.

- <job-ID>, invalid job id

  The number <job-ID> is not a job ID that is associated with any terminal managed by the Human Interface.

- <job-ID>, job does not belong to you

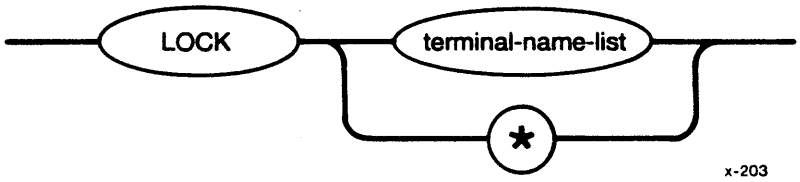
  The user who attempted to delete the interactive job does not have the same user ID as the interactive job or is not the system manager.

- <job-ID>, not deleted
  <job-ID>, <exception value> : <exception mnemonic>

  An exceptional condition occurred, preventing JOBDELETE from deleting the job <job-ID>. JOBDELETE displays the exception code that resulted.

LOCK

This command prevents the Human Interface from automatically recreating the interactive job for a terminal once that interactive job has been deleted. This process is called locking the terminal. The system manager can use this command to lock any terminal. Other users can lock only those terminals whose interactive jobs have the same user ID that they have. The format of this command is as follows:



where:

terminal-name-	One or more terminal device names, separated by commas, of the terminals to be locked. You can obtain the terminal device names by invoking the INITSTATUS command (described earlier in this chapter).
*	A special character indicating that all configured terminals should be locked.

DESCRIPTION

The system manager can use the LOCK command in conjuction with the JOBDELETE command either to selectively delete users from the system or to shut down the entire system. LOCK prevents the Human Interface from recreating a user's interactive job once that job has been deleted. Interactive jobs can be deleted in any of the following ways:

- As a result of the JOBDELETE command (described earlier in this chapter)
- By entering an end-of-file character (CTRL/z) at the terminal

As LOCK locks each terminal, it displays the following message to the user terminal (:CO:):

<terminal-name>, locked

where <terminal-name> is the terminal device name of the locked terminal.

ERROR MESSAGES

- lock not allowed

  You attempted to lock your own terminal. Only system managers can lock their own terminals.

- <terminal-name>, not found

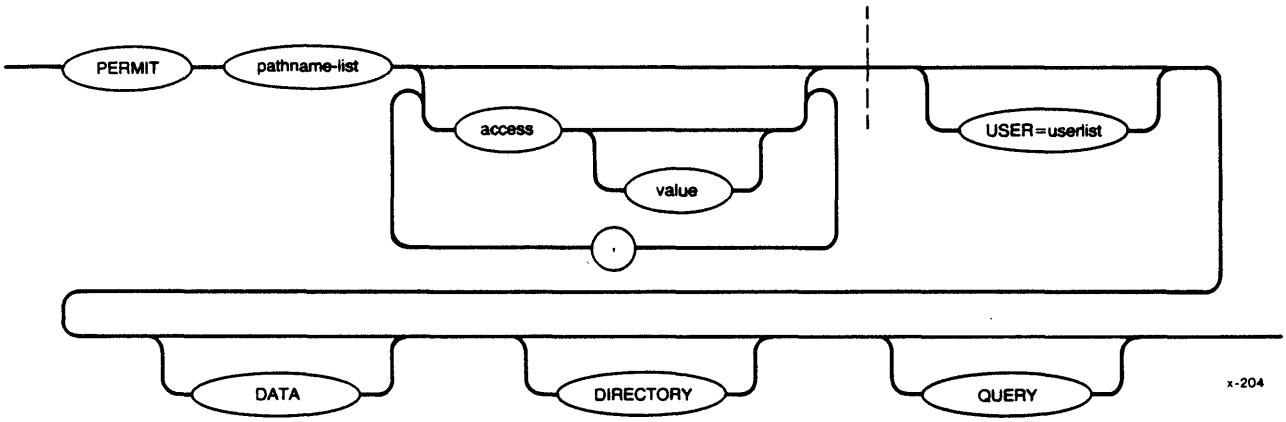
  A terminal with device name <terminal-name> is not configured into your application system.

- not a multi-access system

  The LOCK command does not function if the Human Interface is configured for single-access only.

PERMIT

This command allows you to grant or revoke user access to files that you own. The format of this command is as follows:



INPUT PARAMETERS

pathname-list — One or more pathnames, separated by commas, of the files that are to have their access rights or list of accessors changed.

access — Access characters that grant or rescind the corresponding access to the file, depending on the value parameter that follows. The possible values include:

value	access
D	Delete
L or R	List (for directories) and read (for data files)
A	Add entry (for directories) and append (for data files)
C or U	Change (for directories) and Update (for data files)
N	Rescinds all access not explicitly granted (used without an accompanying value)

If specified without an accompanying value, each access character grants the specified access. Specifying N alone rescinds all access and removes the users specified with the USER parameter from the file's access list. Specifying N with other characters grants the access specified by those characters and rescinds all other access. You can use L and R interchangeably for both data files and directories; likewise C and U.

value

Value which specifies whether to grant or rescind the associated access right. Possible values include:

value	meaning
0	Rescind the access right
1	Grant the access right

The default value is 1. That is, specifying an access character without a value grants the corresponding access.

user-list

User IDs for whom the previously-specified access rights apply. Two special values are also acceptable for this parameter. They are:

WORLD	Special user ID (0FFFFh) giving all users access to the file.
*	Designator indicating that the access rights apply to all users currently in the file's access list.

The Operating System limits each file to three user IDs in the access list. If you omit this parameter, PERMIT assumes the user ID associated with your interactive job.

DATA

Specifies that the access information applies to the data files in the pathname list. If you omit both the DATA and DIRECTORY parameters, PERMIT assumes both.

DIRECTORY

Specifies that the access information applies to the directories in the pathname list. If you omit both the DATA and DIRECTORY parameters, PERMIT assumes both.

QUERY

Causes PERMIT to prompt for permission to modify the access rights associated with each file. It does this by displaying the following message:

<pathname>,
accessor = <new id>, <new access>, PERMIT?--

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

Entry	Action
Y or y	Change the access.
E or e	Exit from the PERMIT command.
R or r	Change the access and continue with the command without further query.
Any other character	Do not change access; continue with PERMIT command and query for next access change, if any.

DESCRIPTION

You can use the PERMIT command to update the access information for the following files:

- Files for which you are listed as the owner.
- Files for which you have change-entry access to the file's directory.

You cannot change the access information for other files. PERMIT can perform the following functions:

- Adding or subtracting users from a file's list of accessors. This list determines which users have access to the file.
- Setting the type of access (access rights) granted to the users in the accessor list.

Currently the Operating System allows only three user IDs in the list of accessors, but one of these IDs can be the special ID WORLD, which grants access to all users.

You specify the type of access to be granted or rescinded by means of access characters and values. You can concatenate access characters and values together or you can separate the individual access specifications with commas. For example, if you want to grant delete access and rescind add and update access, you could enter any of the following combinations:

```
AODUO
AO,D,UO
AOD1UO
AO,D1,UO
```

As you can see from the previous lines, D is equivalent to D1. Also, the order in which you specify access characters is not important.

If there are multiple occurrences of an access character in the PERMIT command, PERMIT uses the last such character to determine the access. For example, the combination:

```
DO,A1,R1,D1
```

is the same as the combination:

```
A1,R1,D1
```

In the first combination, the D1 overrides the DO.

You can use the N character to rescind all access to the file. If specified alone, it removes user IDs from the accessor list. However, the N character can also be useful when changing access rights, if you don't remember the specified user's current access rights. In this case you can specify the N character first, to clear all the access rights, and follow it with other characters to grant the desired access. For example, if you want to grant list access only, instead of specifying:

```
DOAOCOL
```

you could specify:

```
NL
```

After changing the access information for a file, PERMIT displays the following information:

```
<pathname>,
    accessor  = <accessor ID>, <access>
              .
              .
              .
```

where <pathname> is the pathname of the specified file, <accessor ID> is the user ID of one of the files accessors, and <access> indicates the access rights that the corresponding user has. PERMIT displays the access rights as access characters: DLAC for directories and DRAU for data files. If a particular access right is not allowed, the display replaces the corresponding character with a dash (-). For example, the display:

```
-L-C
```

indicates that the corresponding user has list and change access.

ERROR MESSAGES

- <pathname>, accessor limit reached

  The Operating System permits only three IDs in the accessor list of a file. Before you can add another accessor, you must remove one of the current accessors by setting its access rights to N.

- <pathname>, directory CHANGE access required

  Either you are not the owner of the file specified by <pathname>, or you do not have change access to the file's parent directory. You must satisfy one of these two conditions in order to use the PERMIT command.

- <user ID>, duplicate USER control

  You must specify the keyword and parameter combination USER = userlist only once during the PERMIT command. However, you can specify multiple user IDs by separating them with commas in the userlist. PERMIT exits without updating the access rights.

- <character>, invalid access switch

  The character you entered to indicate the access rights for the file was not a valid access character. PERMIT exits without updating the access rights.

- <invalid id>, invalid user id

  The user IDs you supply with the USER parameter must consist of decimal or hexadecimal characters, the characters WORLD, or the character *. PERMIT exits if supplied other characters.

- missing access switches

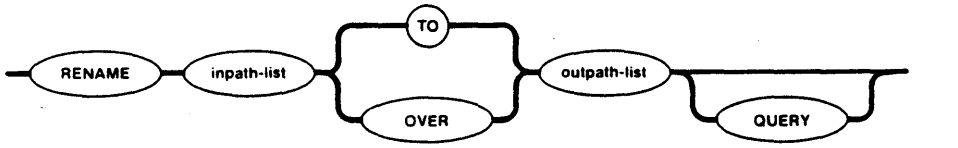
  You must specify one or more access characters with the PERMIT command. PERMIT exits without updating the access rights.

- no files found

  There were no files of the type you specified (data, directory, or both) in the pathname list.

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:



x-321

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 one of the following messages:

<oldname>, rename TO <newname>?

<oldname>, rename OVER <newname>?

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

Entry	Action
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 the output pathname already exists, RENAME displays the following message:

<pathname>, already exists, DELETE?

Enter Y, y, R, or r to delete the existing file. Enter any other character if you do not wish tp delete the file. In the later case, RENAME skips over the specified file without changing it and attempts to rename the next pathname in the list.

OVER — Changes each old pathname in a list to the corresponding new pathname, even if the new pathname already exists. 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 the RENAME command runs, it releases the pathnames of the input files for new uses without performing any further operation on the files.

Another distinction between RENAME and COPY is that RENAME 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 (for example, from one diskette to another). An attempt to do so causes an error message. Use the COPY command or a combination of COPY and DELETE commands if you wish to rename files or move data across volume boundaries.

To use RENAME, you must have delete access to the current file and add-entry access to the destination directory. If you rename a file OVER an existing file, you must also have delete access to the second file.

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/SAMP1 OVER BETA/TEST1      ;allowed
```

NOTE

Changing the name of a directory also changes the path of all files listed in 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

- <old pathname>, DELETE access required

  You cannot rename a file unless you have delete access to that file.

- <new pathname>, directory ADD ENTRY access required

  You cannot rename a file unless you have add-entry access to the destination directory.

- <new pathname>, new pathname same as old pathname

  You specified the same name for the input pathname as you did for the output pathname.

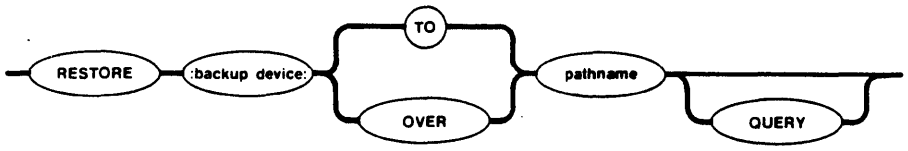
- TO or OVER preposition expected

  Either you used the AFTER preposition with the RENAME command or the number of files in your inpath-list did not match the number in your outpath-list.

RESTORE

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

The format of this command is as follows:



x-322

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:

Entry	Action
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. If a file being restored already exists on the named volume, RESTORE displays the following message:

<pathname>, already exists, OVERWRITE?

Enter one of the following in response to the query:

Entry	Action
Y, y, R, or r	Delete the file and replace it with the one from the backup volume.
E or e	Exit from the RESTORE command.
Any other character	Do not restore the file; go on to the next 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.

Normally, when RESTORE copies files, it copies only those files to which you have access. When it copies these files to the named volume, it establishes your user ID as the owner ID (regardless of what the previous owner ID was). However, if you are the system manager (user ID 0), RESTORE restores all files from the backup volume and leaves the owner ID the same as it was.

When copying files, RESTORE 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.

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. For RESTORE to create new files on a named volume, you must have add entry access to directories on that volume. 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.y
```

where Vx.y 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:

Entry	Action
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 one of the following messages at the Human Interface console output device (:CO:):

<pathname>, restored

or

<pathname>, directory restored

ERROR MESSAGES

- <pathname>, access to directory or file denied

  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 value> : <exception mnemonic>

  <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 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.

- <pathname>, <exception value> : <exception mnemonic>, 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 value> : <exception mnemonic>, 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 value> : <exception mnemonic>

  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 value> : <exception mnemonic>, 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 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. 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.

- <backup device>, invalid backup device

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

- <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.

- <pathname>, 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.

- output specification missing

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

- <pathname>, READ access required

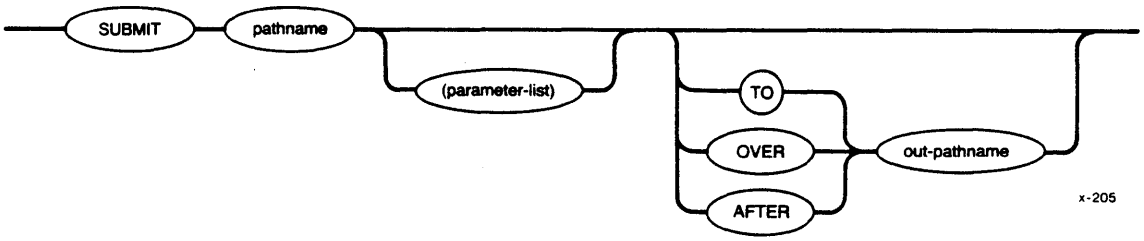
  You do not have read access to a file on the backup volume; therefore RESTORE cannot restore the file.

- <pathname>, too many input pathnames

  You attempted to enter a list of logical names as logical names for backup devices. You can enter only one input logical name per invocation of RESTORE.

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 output file already exists, the SUBMIT command displays the following message:

<pathname>, already exists OVERWRITE?

Enter Y, y, R, or r 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.

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.

out-pathname — Pathname of the file 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 input (:CI:) 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 (each level of SUBMIT requires approximately 10K of dynamic memory). When one nested SUBMIT file completes execution, it returns control to the next higher level of SUBMIT file.

If, during the execution of SUBMIT (or any nested SUBMIT), you enter the CTRL/c character to abort processing, all SUBMIT processing exits and control returns to your user session.

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 (other than :CO:) 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>

ERROR MESSAGES

- <pathname>, end of file reached before end of command

  The last command in the input file was not specified completely. For example, the last line might contain a continuation character.

- <parameter>, incorrectly formed parameter

  You separated the individual parameters in the parameter list with a separator character other than a comma.

- <pathname>, output file same as input file

  You attempted to place the output from SUBMIT into the input file.

- <pathname>, too many input files

  You specified more than one pathname as input to SUBMIT. SUBMIT can process only one file per invocation.

- <parameter>, too many parameters

  You specified more than 10 parameters in your parameter list.

- <pathname>, UPDATE or ADD access required

  SUBMIT cannot write its output to the output file because you do not have update access to the file (if it already exists) or because you do not have add access to the file's parent directory (if the file does not currently exist).

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:MOVE$FILE, 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 :F0:MOVE$FILE (: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 :F0:MOVE$FILE
-
```

SUPER

This command allows operators who are designated as system managers to change their user IDs to the system manager user ID (user ID 0). Having entered the SUPER command, these users can invoke a sub-command to change to any other user ID. The format of this command is as follows:

x-206

DESCRIPTION

SUPER allows you to change your user ID to that of the system manager. It has two sub-commands (CHANGEID and EXIT) that are available only after you have invoked SUPER. CHANGEID allows you to change your user ID to any valid number. EXIT exits the SUPER utility.

In order to invoke SUPER, you must know a password associated with the system manager. This password is stored in the user definition file for user ID 0 (refer to Chapter 6). After you enter the SUPER command, SUPER prompts for the password by displaying:

```
ENTER PASSWORD:
```

You must then enter the correct password. Although the Human Interface doesn't usually distinguish between uppercase and lowercase letters, the password is an exception. It must be exactly the same as the password in the user definition file. (SUPER does not echo your input at the terminal.) After you enter the correct password, SUPER changes your user ID to user ID 0 and issues the following prompt.

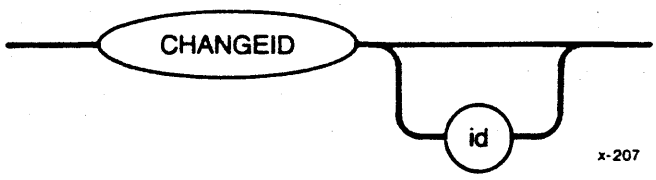
```
super-
```

This prompt is a new system prompt (replacing the "-") that appears whenever the Human Interface is ready to accept input. At this point, you can enter any Human Interface commands and access any files available to the system manager. If you create new files, they will be listed as owned by user ID 0. You can also invoke the sub-commands available with SUPER.

SUBCOMMANDS

There are two sub-commands available with SUPER: CHANGEID and EXIT. You can invoke these sub-commands only after first invoking the SUPER command.

The CHANGEID sub-command allows you to change your current user ID to any value between 0 and 65535 decimal. The format of the CHANGEID sub-command is as follows:



where:

id	Value to which you want to change your user ID. This can be any numeric value from 0 to 65535 decimal, or the characters "WORLD" which specifies ID 65535 decimal. If you omit this value, CHANGEID sets your user ID to that of the system manager (user ID 0).

If you change your user ID to anything other than that of the system manager (user ID 0), the system prompt changes to the following:

```
super(id)-
```

where id is the decimal equivalent of your new user ID (or the characters "WORLD").

The EXIT sub-command exits from the SUPER utility. The format of this sub-command is as follows:

After you enter this sub-command, the Human Interface changes your user ID back to the ID you had before entering the SUPER command. It also changes the system prompt back to the "-" value. To change your user ID again, you must invoke the SUPER command.

ERROR MESSAGES

- <exception value> : <exception mnemonic> cannot set default user

  An internal system problem prevented the Human Interface from changing your user ID.

- <user-id>, invalid user id

  The user ID you specified contained invalid characters or was not in the range 0 to 65535 decimal.

- invalid password

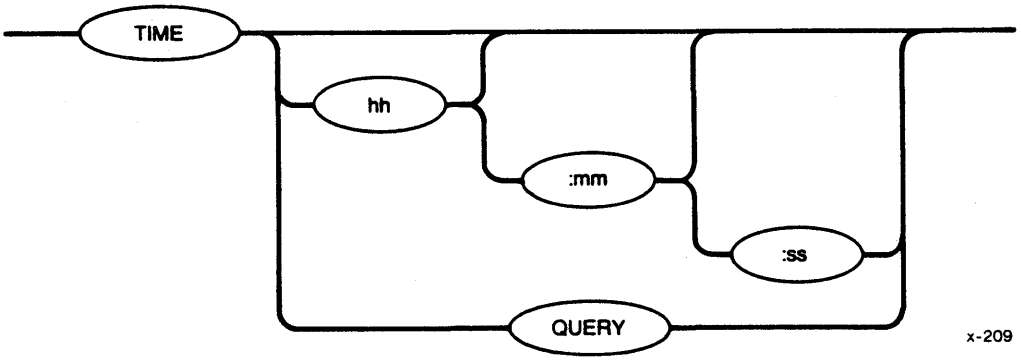
  The password you entered does not match the password associated with the system manager that is listed in the user definition file.

- <exception value> : <exception mnemonic>, SUPER is un-available

  The Human Interface encountered an error while reading the password you entered or while accessing the system manager's user definition file (to determine if the password is correct). This message lists the exception code that occurs.

TIME

This command sets the system clock. If no new time is entered, the TIME displays the current system date and time. The format is as follows:



INPUT PARAMETERS

hh	Hours specified as 0 through 24.
mm	Minutes specified as 0 through 59. If you omit this parameter, 0 is assumed.
ss	Seconds specified as 0 through 59. If you omit this parameter, 0 is assumed.
QUERY	Causes TIME to prompt you for the time by issuing the following message: TIME: TIME continues to issue this message until you enter a valid time.

DESCRIPTION

You must separate the individual time parameters with colons.

If you omit the time parameters, TIME displays the current date and time in the following format:

dd mmm yy, hh:mm:ss

where dd mmm yy indicates the date and hh:mm:ss indicates the time.

In order to obtain the correct time when you enter the TIME command without parameters, you must initially set the time. If you request the time on a system in which you haven't already set the time (or on a non-timing system), 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.

ERROR MESSAGES

- <time>, invalid time

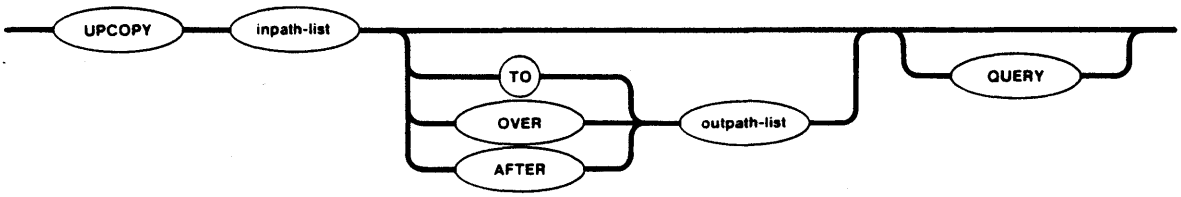
  You specified an invalid or out-of-range entry for one or more of the time parameters.

- <parameter>, invalid syntax

  You specified both a time and the QUERY parameter in the TIME command.

UPCOPY

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



x-323

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:

<in-pathname>, copy up TO <out-pathname>?

<in-pathname>, copy up OVER <out-pathname>?

<in-pathname>, copy up AFTER <out-pathname>?

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

Entry	Action
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. If the output file already exists, UPCOPY displays the the following message: <pathname>, already exists, OVERWRITE? Enter Y, y, R, or r if you wish to write over the existing file. Enter any other character if you do not wish the file to be overwritten. 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 input files 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 via the iSBC 957B package, and the package must be running. To do this, you must start your iRMX 86 system from the development system terminal (either by loading the software into the target system and using the monitor G command to start execution, or by using the monitor B command to bootstrap load the software). UPCOPY does not function if you start up your system from the iRMX 86 terminal or if you establish the link between development system and target system after starting up your iRMX 86 system.

The user ID of the user who invokes the UPCOPY command is considered the owner of new files created by UPCOPY. Only the owner can change the access rights associated with the file (refer to the PERMIT command).

As it copies each ISIS-II file in the input list, UPCOPY displays one of the following messages at the terminal, as appropriate:

<in-pathname>, copied up TO <out-pathname>

<in-pathname>, copied up OVER <out-pathname>

<in-pathname>, copied up AFTER <out-pathname>

When the UPCOPY command is executing, the iSBC 957B package disables interrupts. This affects services such as the time-of-day clock. Also, the Operating System is unable to receive any characters that you type-ahead while the UPCOPY command is executing.

ERROR MESSAGES

- <pathname>, ISIS ERROR: <nnn>

  An ISIS-II Operating System error occurred when UPCOPY tried to transfer the file to the Microcomputer Development System. Refer to the INTELLEC SERIES III MICROCOMPUTER DEVELOPMENT SYSTEM CONSOLE OPERATING INSTRUCTIONS for a description of the resulting error code.

- ISIS link not present

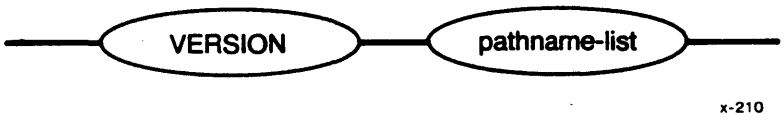
  The the iRMX 86 system is not connected to the development system via the iSBC 957B package.

- <pathname>, UPDATE or ADD access required

  Either you cannot overwrite the information in a file because you do not have update access to it, or you cannot copy information to a new file because you do not have add entry access to the file's parent directory.

VERSION

This command displays the version number of a command or other program, if that command has a version number encoded in its object code. The format of this command is as follows:



x-210

INPUT PARAMETER

pathname-list	One or more pathnames, separated by commas, of commands for which a version number is desired.

DESCRIPTION

When you enter the VERSION command, it displays the version number of each command, if there is one, in the following format:

<pathname>, <command-name> version is x.y

where:

<pathname>	Pathname of the file containing the command.
<command-name>	Name of the specified command; Intel-supplied commands have names as listed in this manual.
xxxx	Version number of the command.

You can use VERSION to determine the version number of any Human Interface command. You can also use it to determine the version numbers of commands that you write. However, for VERSION to work on your commands, you must include a literal string in the command's source code to specify the name of the command and its version. The string must contain the following information:

```
'command_version_number=xxxx',
'command_name=yyyy...yyy',0
```

where:

command_version_number=	You must specify this portion exactly as shown (lower case, underscore separating the words, no spaces).

xxxx	Version number of the product. This can be any four characters, but it must be exactly four characters long.
command_name=	This portion is optional. However, if you want VERSION to recognize and display the program name, you must specify this portion exactly as shown.
yyyy...yyy	Name of the command. This can be any number of characters.
0	The literal string must be terminated with a byte of binary zero.

An example of such a literal string is:

```
DECLARE version (*)  BYTE  DATA('program_version_number=V8.5',
                               'program_name=MYPROGRAM',0);
```

If your program includes this declaration, when you invoke VERSION, it will display the following information:

```
<pathname>, MYPROG version is V8.5
```

A literal string that does not include the program name is:

```
DECLARE vers2(*)   BYTE  DATA('program_version_number=1983',0);
```

If your program includes this declaration, when you invoke VERSION, it will display the following information:

```
<pathname>, version is 1983
```


***

CHAPTER 4. UDI SYSTEM CALLS

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 provides a standard interface by 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. This section is the heart of the 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 functional groupings.

- EXAMPLE PROGRAM. At the end of the chapter is a sample program using the UDI system calls.

USING THE UDI

This section contains information about:

- Exceptional conditions that can occur when you use UDI system calls

- UDI Libraries and INCLUDE files

- Special data types referred to in descriptions of UDI system calls

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.

- 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.

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 LARGE.LIB, COMPAC.LIB, and SMALL.LIB. If your program corresponds to the LARGE or MEDIUM models of segmentation, link it to LARGE.LIB. If your program corresponds to the SMALL or COMPACT models of segmentation, link it to SMALL.LIB or COMPAC.LIB, respectively.

This chapter will assume that the libraries have been transferred to the directory :SD:UDI (from the Include Files Diskette delivered with the iRMX 86 PC product). The pathname for the COMPACT library, for example, is :SD:UDI/COMPAC.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. Each UDI system call has a corresponding file on the Include File Diskette sent by Intel (see Preface); the file contains a PL/M-86 EXTERNAL PROCEDURE statement for that system call. You can build a single file to INCLUDE the files that are for the system calls used in your programs.

All Universal Development Interface (UDI) System Call Declarations are contained in the following files. The name of the system call is listed first, followed by the name of the file containing the external declaration for the call. The list is alphabetical by system call name, and is printed in groups of five for readability.

ALLOCATE	UALLOC.EXT
ATTACH	UATACH.EXT
CHANGE$ACCESS	UCHAC.EXT
CHANGE$EXTENSION	UCHEXT.EXT
CLOSE	UCLOSE.EXT
CREATE	UCREAT.EXT
DECODE$EXCEPTION	UDCEX.EXT
DECODE$TIME	UDCTIM.EXT
DELETE	UDELET.EXT
DETACH	UDTACH.EXT
EXIT	UEXIT.EXT
FILE$INFO	UFLINF.EXT
FREE	UFREE.EXT
GET$ARGUMENT	UGTARG.EXT
GET$CONNECTION$STATUS	UGTCN.EXT
GET$EXCEPTION$HANDLER	UGTEXH.EXT
GET$SIZE	UGTSIZ.EXT
GET$SYSTEM$ID	UGTSID.EXT
GET$TIME	UGTTIM.EXT
OPEN	UOPEN.EXT
OVERLAY	UOVLY.EXT
READ	UREAD.EXT
RENAME	URENAM.EXT
RESERVEIOMEMORY	URSIOM.EXT
SEEK	USEEK.EXT
SPECIAL	USPECL.EXT
SWITCH$BUFFER	UTRUNC.EXT
TRAP$CC	UTRAPC.EXT
TRAP$EXCEPTION	UTRPEX.EXT
WRITE	UWRITE.EXT

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.
DWORD	A 32-bit integer.
STRING	A sequence of bytes, the first of which contains the number of characters in the STRING.
POINTER	Equivalent to PL/M-86 type POINTER. It is two bytes under the small model of segmentation; four bytes in other cases.
SELECTOR	A 16-bit iAPX 86,88 paragraph number (the base portion of a four-byte pointer).
TOKEN	A value passed between a program and the Operating System to represent an object. You can declare TOKEN to be a SELECTOR if your compiler supports the SELECTOR data type; otherwise declare it to be a WORD.

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 PL/M-86 invocation of the system call (as opposed to formal invocation) and an ASM-86 calling sequence. These examples are shown 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 DQGETSIZE to receive information about an allocated memory segment.

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.

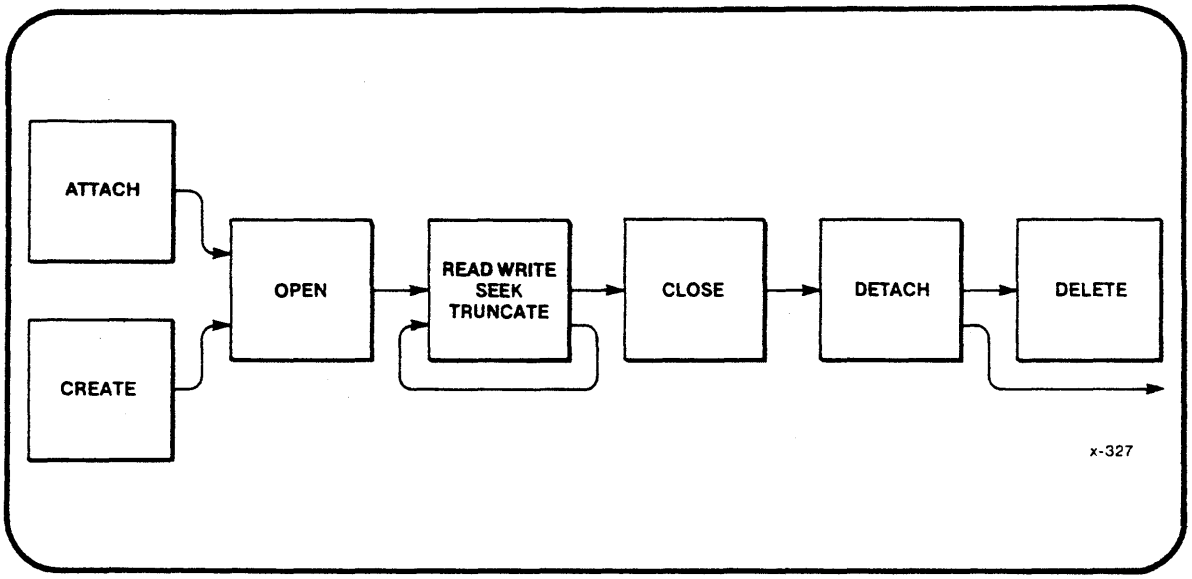


Figure 4-1. Chronology Of System Calls

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.

This section describes the major File Handling system calls. Refer the System Call Dictionary for a complete list.

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.
- DQGETEXCEPTION$HANDLER, which is an informative system call returning the address of the current exception handler: either the default system handler, or one you specify with DQ$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.

SYSTEM CALLS

This section lists contains descriptions of every UDI system call. Table 4-1 contains lists the calls by functional classes, and includes, for each call:

- The name of the call
- A brief description of the call
- The page number of the description

Table 4-1. System Call Dictionary

SYSTEM CALL	FUNCTION PERFORMED	PAGE
	PROGRAM CONTROL CALLS	
DQ$EXIT	Exits from the current application job.	4-23
DQ$OVERLAY	Causes the specified overlay to be loaded.	4-37
DQ$TRAP$CC	Captures control when CTRL/c is typed.	4-48
	FILE-HANDLING CALLS	
DQ$ATTACH	Creates a connection to a specified file.	4-12
DQ$CHANGE$-ACCESS	Changes the access rights associated with a file or directory.	4-13
DQ$CHANGE$-EXTENSION	Changes the extension of a file name in memory.	4-15
DQ$CLOSE	Closes the specified file connection.	4-16
DQ$CREATE	Creates a file for use by the application.	4-17
DQ$DELETE	Deletes a file.	4-21
DQ$DETACH	Closes a file and deletes its connection.	4-22
DQGETCON-NECTION$STATUS	Returns status of a file connection.	4-29
DQ$FILE$INFO	Returns data about a file connection.	4-24
DQ$OPEN	Opens a file for a particular type of access.	4-35
DQ$READ	Reads the next sequence of bytes from a file.	4-39
DQ$RENAME	Renames the specified file.	4-41
DQ$SEEK	Moves the current position pointer of a file.	4-43
DQ$SPECIAL	Sets terminal line-edit/transparent mode.	4-45
DQ$TRUNCATE	Truncates a file to the specified length.	4-50
DQ$WRITE	Writes a sequence of bytes to a file.	4-51

Table 4-1. System Call Dictionary (continued)

SYSTEM CALL	FUNCTION PERFORMED	PAGE
	MEMORY MANAGEMENT CALLS	
DQ$ALLOCATE	Requests a memory segment of a specified size.	4-10
DQ$FREE	Returns a memory segment to the system.	4-26
DQGETSIZE	Returns the size of the specified segment.	4-32
DQ$RESERVE$-IO$MEMORY	Requests that memory be set aside for I/O operations overhead.	4-42
	EXCEPTION-HANDLING CALLS	
DQ$DECODE$-EXCEPTION	Converts an exception numeric code into its equivalent mnemonic.	4-18
DQGETEXCEPT-ION$HANDLER	Returns a POINTER to the address of the program currently being used to process errors.	4-31
DQ$TRAP$-EXCEPTION	Identifies a custom exception processing program for a particular type of error.	4-49
	UTILITY AND COMMAND PARSING	
DQ$DECODE$TIME	Returns system time and date in binary and in ASCII character format.	4-19
DQGETARGUMENT	Returns an argument from a STRING.	4-27
DQGET-SYSTEM$ID	Returns the name of the underlying operating system supporting the UDI.	4-33
DQGETTIME	(Obsolete: included for compatability.)	4-34
DQ$SWITCH$BUFFER	Selects a new buffer from which to process commands.	4-47

DQ$ALLOCATE

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

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

INPUT PARAMETER

size	A WORD which, • 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$addr	A SELECTOR in which the Operating System places the base address of the memory segment. If the request fails because the memory requested is not available, this argument will be 0FFFFH, and the system will return an E$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.

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, (or SELECTOR)
          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    DQALLOCATE
                    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 the POINTER offset.

DQ$ATTACH

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	The TOKEN for the connection to the file; returned by the iRMX 86 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 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.

DQ$CHANGE$ACCESS

The DQ$CHANGE$ACCESS allows you to change the access rights of the owner of a file, or of WORLD.

```
CALL DQ$CHANGE$ACCESS (path$ptr, user, access, except$ptr);
```

INPUT PARAMETERS

path$ptr — A POINTER to a STRING containing the pathname of the file.

user — A BYTE specifying the type of user whose access is to be changed:

Value	Meaning
zero	owner of the file
one	WORLD (all users on the system)
2 -255	reserved by Intel

access — A BYTE specifying the type of access to be granted to the user. The flags in this word are encoded as follows. (Bit 0 is the low-order bit.)

Bit	Meaning
0	User can delete the file or directory
1	Read (file) or List (directory)
2	Append (file) or Add entry (directory)
3	Update (read and write: file) or Change Access (directory)
4-7	should be 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

You can use this system call to change the access rights of a file. You must have Change Access rights to the directory in which the file or directory is contained. This call affects only connections made after the call is issued, but does not affect current connections to the file.

DQ$CHANGE$EXTENSION

DQ$CHANGE$EXTENSION changes or adds the extension at the end of a file name stored in memory (it doesn't affect the file name on the mass storage volume).

```
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 to the file name in memory. 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 DQGETARGUMENT 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).

DQ$CLOSE

DQ$CLOSE 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 DQ$CLOSE (connection, except$ptr);
```

INPUT PARAMETER

connection	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 DQ$CLOSE system call closes a connection that has been opened by the DQ$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

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	The TOKEN for the connection to the file; returned by the iRMX 86 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 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

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$DECODE$TIME

DQ$DECODE$TIME returns the current system time and date as a Double Word integer and as a series of ASCII character bytes.

```
CALL  DQ$DECODE$TIME (time$ptr, except$ptr);
```

INPUT PARAMETER

time$ptr — A POINTER to a buffer of the following structure:

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

DATE and TIME are returned by this system call, as described below. If SYSTEM$TIME is not zero when you call DQ$DECODE$TIME, it is converted (decoded) to a series of ASCII bytes representing the date and time.

If SYSTEM$TIME is zero, the current system clock time (number of seconds since January 1, 1978) is first returned and then decoded into DATE and TIME.

OUTPUT PARAMETERS

time$ptr — The buffer described above is used to return either:

- The current system time as a DWORD integer, and as a series of ASCII bytes decoded from the DWORD value.
- The ASCII bytes representing the value you passed in the SYSTEM$TIME 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 returns the current date and time, each as a series of bytes (note that this is not a STRING).

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

If, when you call DQ$DECODE$TIME, the SYSTEM$TIME parameter is zero, the call will first get the system time (number of seconds since January 1, 1978) and then decode it into the series of bytes described above.

But if SYSTEM$TIME is not zero, the system call will simply convert it to the series of bytes. You can use the system call DQ$FILE$INFO to get two DWORD values associated with a file (the last time the file was updated, the time the file was created) and use DQ$DECODE$TIME to convert the dates to a series of bytes.

DQ$DELETE

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

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

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

INPUT PARAMETER

connection	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.

DQ$EXIT

DQ$EXIT returns control from your program to the Operating System.

```
CALL DQ$EXIT (end$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:

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

DESCRIPTION

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

Calling DQ$EXIT cannot result in an exception code.

DQ$FILE$INFO

DQ$FILE$INFO returns information about a particular file.

```
CALL  DQ$FILE$INFO (connection, mode, file$info$ptr, except$ptr);
```

INPUT PARAMETERS

connection — A WORD representing the connection for the file.

mode — A BYTE specifying whether to return the User ID of the owner of the file. Set as follows:

Value	Meaning
0	do not return owner's User ID.
1	Return the owner's User ID.
2-255	Reserved by Intel

OUTPUT PARAMETERS

file$info$ptr — A POINTER to a buffer into which the Operating System returns the information requested. The structure of this buffer is:

```
DECLARE FDATA STRUCTURE
    (OWNER(15)          STRING,
    LENGTH              DWORD,
    TYPE                BYTE,
    OWNER$ACCESS        BYTE,
    WORLD$ACCESS        BYTE,
    CREATE$TIME         DWORD,
    LAST$MOD$TIME       DWORD,
    RESERVED(20)        BYTE);
```

where:

OWNER — A STRING containing the User ID of the file owner.

TYPE — A number indicating the type of file, as follows:

0 data file
1 directory

OWNER$ACCESS — A BYTE specifying the type of access granted to the owner. The flags in this word are encoded as follows. (Bit 0 is the low-order bit.)

Bit	Meaning
0	Delete
1	Read (data file) or Display (directory)
2	Append (data file) or Add Entry (directory)
3	Update (read and write: file) or Change Access (directory)

WORLD$ACCESS — A BYTE specifying the type of access granted to the WORLD (all users on the system). The flags in this word are encoded as follows. (Bit 0 is the low-order bit.)

Bit	Meaning
0	Delete
1	Read (data file) or Display (directory)
2	Write (data file) or Add Entry (directory)
3	Update (read and write: file) or Change Access (directory)

CREATE$TIME — The date and time that the file was created, expressed as the number of seconds since Jan. 1, 1978.

(You can convert this date/time to ASCII characters, use the system call DQ$DECODE$TIME.)

LASTMODTIME — The date and time that the file or directory was last modified. For data files, modified means written or truncated; for directories, modified means and entry was changed or an entry was added.

(You can convert this date/time to ASCII characters, use the system call DQ$DECODE$TIME.)

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

DESCRIPTION

The DQ$FILE$INFO allows a program to obtain information about a file or a directory.

DQ$FREE

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$addr	A TOKEN for the base address of the segment that is to be deleted. This is the base address returned to your program by DQ$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.

DQGETARGUMENT

The DQGETARGUMENT 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 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 DQGETARGUMENT, 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 ( ) and any characters with hexadecimal values between 0 and 20H.

The Operating System strips out ampersands (&) and semicolons (;).

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

DQGETCONNECTION$STATUS

The DQGETCONNECTION$STATUS system call returns information about a file connection.

```
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$ptr — A POINTER to a structure in which the Operating System will place the status information. The structure pointed to by info$ptr should be:

```
DECLARE INFO STRUCTURE
            (OPEN          BYTE,
             ACCESS        BYTE,
             SEEK          BYTE,
             FILE$PTR$     DWORD;
```

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.

BIT	ACCESS
0	delete
1	read
2	write
3	update (read and write)

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 — This Double Word integer marks 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 seek is not supported by the device (for example, seek operations are not valid for a line printer.)

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

DESCRIPTION

DQGETCONNECTION$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.

DQGETEXCEPTION$HANDLER

DQGETEXCEPTION$HANDLER returns the address of the current exception handler.

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

OUTPUT PARAMETERS

address$ptr	A POINTER to a POINTER in which the Operating System returns the entry point of the current exception handler.
except$ptr	A POINTER to a WORD where the system places the condition code. Condition codes are described in Appendix A.

DESCRIPTION

DQGETEXCEPTION$HANDLER is an system call that returns to your program the address of the current exception handler. This is the address specified in the last call to DQ$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 two-word pointer, even if called from a program compiled under the SMALL model of segmentation.

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

DQGETSIZE

DQGETSIZE returns the size of an allocated memory segment.

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

INPUT PARAMETER

base$addr	A TOKEN (WORD or SELECTOR) for the base address of a segment of memory that was allocated with the DQ$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 DQGETSIZE.

DQGETSYSTEM$ID

DQGETSYSTEM$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

DQGETTIME

DQGETTIME returns the current date and time in character format.

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

This system call is included here for compatability with previous versions of the UDI. You should use the system call DQ$DECODE$TIME for this function.

DQ$OPEN

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 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:

Value	Meaning
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 as many connections open at one time as memory allows.

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.

DQ$OVERLAY

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 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 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 system device (:SD:) in the directory UTILS 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,      &
**:SD:UDI/LARGE.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)
```

DQ$READ

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 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.

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.

DQ$RENAME

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 pathname 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$RESERVE$IO$MEMORY

The DQ$RESERVE$IO$MEMORY allows you to reserve enough memory to ensure that you will be able to OPEN and ATTACH the files that this program uses.

```
CALL DQ$RESERVE$IO$MEMORY (number$files, number$buffers, except$ptr);
```

INPUT PARAMETERS

number$files	The number of files that you expect the program will have attached simultaneously.
number$buffers	The total number of buffers that will be needed at one time.

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$RESERVE$IO$MEMORY ensures that, as your program requests memory segments (using DQ$ALLOCATE), you will not use up memory that you will later need for attaching and opening files. For example, if your program will attach and open two files at the same time, each of which has two buffers, (specified when the DQ$OPEN call is issued), number$files should be two and number$buffers four.

Your program should issue DQ$RESERVE$IO$MEMORY before the first DQ$ALLOCATE system call is issued.

If you issue this call more than once in a program, it simply changes the amount of memory reserved.

RESTRICTION

This system call is effective as long as you use only UDI system calls to communicate with the iRMX 86 Operating System.

DQ$SEEK

DQ$SEEK changes the file position pointer.

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

INPUT PARAMETERS

connection — A TOKEN for an open connection whose file pointer you wish to move.

move$count — A DWORD (Double Word) integer 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:

Mode	Meaning
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 the 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 non-sequential 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 not written to will contain undefined information.

DQ$SPECIAL

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

```
CALL DQ$SPECIAL (mode, conn$ptr, except$ptr);
```

INPUT PARAMETERS

mode — A BYTE used to change the mode of terminal input. The values and their meanings are:

Value	Meaning
1	Transparent
2	Line editing
3	Immediate Transparent

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. Unless you have issued this system call since the system was bootstrap loaded, the mode will be line editing. But by using DQ$SPECIAL 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.

Value	Meaning
1	Transparent. 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.
2	Line Editing. This means that the console operator has the opportunity to correct typing errors with special keys (like RUBOUT) before the application program receives the characters typed. Line editing characters and their effects are described in Chapter 2.
3	Immediate Transparent. This is nearly the same as Transparent 1 mode, except that in Transparent 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 the "type-ahead" buffer.

DQ$SWITCH$BUFFER

DQ$SWITCH$BUFFER is used with DQGETARGUMENT 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 an 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 DQGETARGUMENT. Each call to DQGETARGUMENT fetches an argument from the line pointed to by buff$ptr.

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

You can use DQ$SWITCH$BUFFER any number of times to point to different strings in your program. However, you cannot use DQ$SWITCH$BUFFER to return to the command line that invoked the program. So you should use DQGETARGUMENT to parse all elements of the command line before issuing the first call to DQ$SWITCH$BUFFER.

DQ$TRAP$CC

The DQ$TRAP$CC allows you to specify a procedure (handler) to be entered if a user types CTRL/c at the keyboard terminal.

```
CALL DQ$TRAP$CC (entry$pnt, except$ptr);
```

INPUT PARAMETERS

entry$pnt	A POINTER to the entry point of your CTRL/C handler.

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$TRAP$CC is used if you do not want the Operating System to abort the current program that is being run from the the terminal (:CI:) when the user types CTRL/c.

For example, if you have a special message that you want to display on the users terminal, or if you want the Control/c character to have a meaning different than the normal one (see the description of CTRL/c in Chapter 2), you can use this system call to identify the procedure to which to transfer control when CTRL/c is typed.

DQ$TRAP$EXCEPTION

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	A POINTER to a POINTER containing the entry point of the alternate exception handler.

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

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 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.

DQ$WRITE

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 buffer containing 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.

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 (an 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.

EXAMPLE PROGRAM

This program provides an example of UDI system calls.

After the program listing we have listed the compiler and linker commands used to build for program, and a listing of the linker map.

```
$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 data are 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;

/* These include files are described at the beginning of the chapter */

$include(:include:ltksel.lit)
$include(:include:uexit.ext)
$include(:include:uclose.ext)
$include(:include:uwrite.ext)
$include(:include:uread.ext)
$include(:include:uopen.ext)
$include(:include:ucreat.ext)
$include(:include:ugtarg.ext)
$include(:include:uatach.ext)
$include(:include:udcex.ext)

DECLARE
    CR      LITERALLY '0DH',
    LF      LITERALLY '0AH',
    E$OK    LITERALLY '0';

DECLARE
    co$conn      TOKEN;
```

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;

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


    DECLARE st WORD;

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

EXAMPLE PROGRAM (continued)

```
    DECLARE
        buffer(1024) BYTE,
        in$bp        POINTER,
        in$char      BASED in$bp BYTE,
        nextchar     BASED in$bp (2) BYTE,
        in$count     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 (the first argument).
     *.....................................................................
     */

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



    /*.....................................................................
     *  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);
```

EXAMPLE PROGRAM (continued)

```
    /*.....................................................................
     *  Find out if there is an output file specified.  If so, attach
     *  and open it.  If not, use :CO: for output.
     *.....................................................................
     */

    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;

    /*.....................................................................
     *  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:
```

EXAMPLE PROGRAM (continued)

```
    /*.....................................................................
     *  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;

    /*.....................................................................
     */
```

The program UPPER was compiled and built with the following commands:

```
    plm86 upper.p86
    link86 upper.obj, :lib:compac.lib  to upper  bind mempool(5000H)
```

The linker map is on the next page.

```
iRMX 86 8086 LINKER, V2.0

INPUT FILES: UPPER.OBJ, :LIB:COMPAC.LIB
OUTPUT FILE: UPPER
CONTROLS SPECIFIED IN INVOCATION COMMAND:
  BIND MEMPOOL(5000H)
DATE:  14/02/82  TIME:  12:05:37

LINK MAP OF MODULE UPPER

LOGICAL SEGMENTS INCLUDED:
LENGTH ADDRESS ALIGN  SEGMENT          CLASS           OVERLAY
 02F6H ------    W    CODE             CODE
 001EH ------    W    CONST            CONST
 0475H ------    W    DATA             DATA
 0454H ------    W    STACK            STACK
 0000H ------    W    MEMORY           MEMORY
 0000H ------    G    ??SEG

INPUT MODULES INCLUDED:
UPPER.OBJ(UPPER)
:LIB:COMPAC.LIB(DQATTACH)
:LIB:COMPAC.LIB(DQCLOSE)
:LIB:COMPAC.LIB(DQCREATE)
:LIB:COMPAC.LIB(DQDECODEEXCEPTION)
:LIB:COMPAC.LIB(DQEXIT)
:LIB:COMPAC.LIB(DQGETARGUMENT)
:LIB:COMPAC.LIB(DQOPEN)
:LIB:COMPAC.LIB(DQREAD)
:LIB:COMPAC.LIB(DQWRITE)
:LIB:COMPAC.LIB(SYSTEMSTACK)


GROUP MAP

GROUP NAME:  CGROUP
   OFFSET  SEGMENT NAME
   0000H   CODE

GROUP NAME:  DGROUP
   OFFSET  SEGMENT NAME
   0000H   CONST
   001EH   DATA

SYMBOL TABLE OF MODULE UPPER

BASE    OFFSET TYPE SYMBOL              BASE    OFFSET TYPE SYMBOL

G(1)    0293H  PUB  DQATTACH            G(1)    029EH  PUB  DQCLOSE
G(1)    02A9H  PUB  DQCREATE            G(1)    02B4H  PUB  DQDECODEEXCEPTION
G(1)    02BFH  PUB  DQEXIT              G(1)    02CAH  PUB  DQGETARGUMENT
G(1)    02D5H  PUB  DQOPEN              G(1)    02E0H  PUB  DQREAD
G(1)    02EBH  PUB  DQWRITE             S(4)    006CH  PUB  SYSTEMSTACK
                              ***
```

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 on the hardware described here.

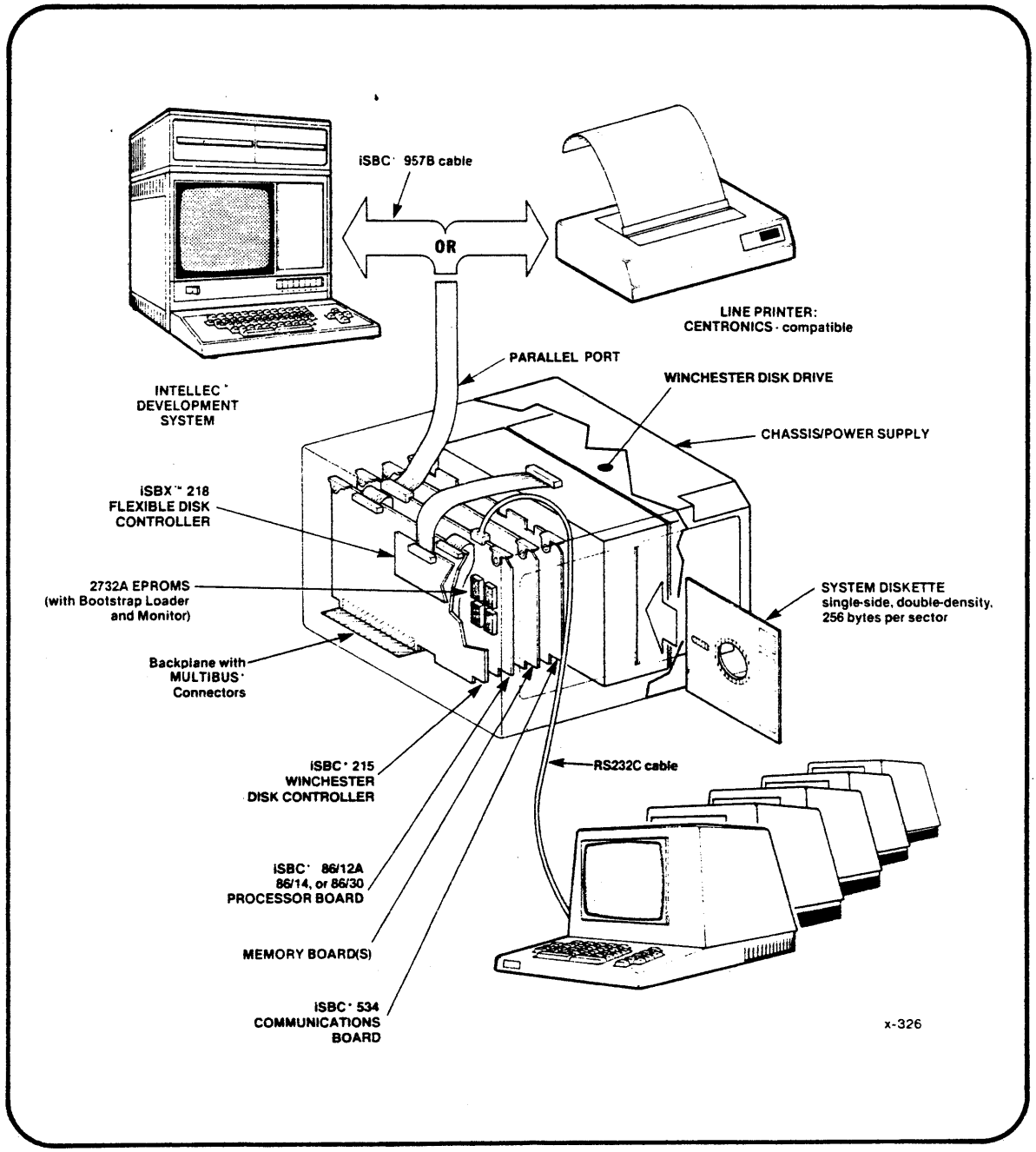


Figure 5-1. The iRMX™ 86 PC Hardware

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. You can remove the page and refer to it as you work on the devices. These charts are the last pages in this chapter.

THE iRMX™ 86 PC HARDWARE ENVIRONMENT

To use the iRMX 86 PC Operating System, some hardware is required, and some is optional.

REQUIRED HARDWARE

The Preconfigured iRMX 86 Operating System requires the following hardware:

- Single-Board Computer. An Intel iSBC 86/12A, iSBC 86/14, or iSBC 86/30 Single Board Computer.
- Flexible Disk Drive. A flexible diskette controller with at least one 8-inch drive.

  This disk drive should conform to the size and recording density of the diskette on which you receive the Preconfigured iRMX 86 Operating System (8-inch, single-sided, double-density, 256 bytes per sector).

  Although you can boot and run the system with one flexible diskette drive, you will need at least one other disk drive to do useful work with the system.
- Keyboard Terminal. A keyboard terminal connected to the serial line on your single-board computer.
- Chassis. An appropriate chassis/cardcage/power-supply unit.
- Memory. At least 256 K-bytes.

OPTIONAL HARDWARE

You can include the following optional hardware in your system:

- Four More Terminals. An iSBC 534 Four Channel Communications Expansion Board with one to four keyboard terminals.

- Winchester Disk. A Winchester hard disk drive connected to an iSBC 215 Disk Controller.

- Total of Eight Flexible Disk Drives. Up to four flexible diskettes connected to an iSBC 208 Flexible Disk Controller and up to four flexible disk drives connected to an iSBX 218 Flexible Disk Controller Multimodule. You can use the iSBC 218 Multimodule only if you also have an iSBC 215 Disk Controller.

  There are some restrictions about size, density, number-of-sides, and bytes-per-sector of these drives. These restrictions are explained later.

- Line Printer or Microcomputer Development System. Either a line printer or an iSBC 957B hardware/software package connected through the parallel port on your Single-Board Computer. 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 Intel Single Board Computers: iSBC 86/12A, iSBC 86/14, or iSBC 86/30. The characteristics of these computers are summarized in Table 5-1.

Table 5-1. Single Board Computers

Single Board Computer	On-board Memory	Multimodule™ Memory (optional)	Clock Frequency
iSBC® 86/12A	32 K-bytes	64 K-bytes	5MHZ
iSBC® 86/14	32 K-bytes	64 K-bytes	5 or 8 MHZ
iSBC® 86/30	128 K-bytes	256 K-bytes	5 or 8 MHZ

FLEXIBLE DISKETTE CONTROLLERS AND DRIVES

You can connect flexible disk drives to your system using an iSBC 208 Controller Board, an iSBC 218 Multimodule Controller Board, or both. If you use the iSBC 218 Controller, it must be mounted on an iSBC 215 Winchester Controller Board. The following applies to the iSBC 208 and to the iSBC 218 Controllers:

- It 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 controller will handle up to four drives. The third and fourth drives on a controller must be 8-inch, standard format drives. (Standard-format is explained in the note below.)
- The system uses soft-sectored diskettes, which means that you must format new diskettes using the FORMAT command described in Chapter 3.

NOTE

The iRMX 86 PC System is delivered on standard format diskettes: 8-inch double-density, single-sided diskettes having 256 bytes-per-sector.

Physical names for iRMX 86 PC disk drives, including Winchester drives, are listed in Chapter 3 in the description of the ATTACHDEVICE command. Chapter 6, SYSTEM MANAGEMENT, describes how to make a copy of the System Diskette.

WINCHESTER DISK DRIVE

You can connect one Winchester Disk to your system using the iSBC 215 Disk Controller Board. (This also allows you to use the iSBC 218 Flexible Disk Controller described in the previous section.) You have a choice of four types of drives.

- Priam 3450: approximately 35 MB capacity
- Pertec D8000: approximately 20 MB
- Memorex 101: approximately 10 MB
- Shugart SA1002: approximately 5 MB (you can use an SA1004, but only 5 MB of the disk will be used).

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.

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

Notes: 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

ADDITIONAL TERMINALS

You can connect up to four keyboard terminals, using the serial ports on an iSBC 534 Communication Expansion Controller. If you connect less than four terminals to the board, use the ports in ascending order, starting at Port 0 (zero). For example, if you use only two terminals, use Ports 0 and 1.

MEMORY

The iRMX 86 PC Operating System requires at least 256K bytes of memory. In addition, you will need sufficient memory to run 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.

iSBC® 957B PACKAGE

You can use the parallel port for copying files to and from an Intellec 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 gives some hints about installing boards in a Multibus backplane, and describes how to modify the following boards:

- iSBC 208 and iSBX 218 Flexible Diskette Controllers
- iSBC 215 Generic Winchester Controller Board
- iSBC 86/12A, iSBC 86/14, and iSBC 86/30 Single Board Computers
- iSBC 534 Communications Expansion Board

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 you start with boards having only 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 the board has a RAM expansion module.
- Whether you are using serial bus priority resolution (the factory default in all cases) or parallel priority resolution.

The instructions for the individual boards 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.

HINTS ABOUT THE MULTIBUS®

When installing boards in the Multibus backplane, you must be sure that bus master boards are set up properly to resolve bus priority, and you should take precautions to avoid bus noise problems. Here are brief helps regarding each, and a recommendation for a booklet that explains more about each.

Bus Priority Resolution

Each Multibus master board contains hardware that automatically resolves bus priority. There are two techniques that this hardware can use: serial resolution or parallel resolution. When you put together your hardware system, you must ensure that all master boards in your system use the same technique.

If you fail to do this, the individual boards will be able to perform on-board operations correctly, but will be unable to perform operations that require more than one board. Two symptoms are being able to access on-board memory but not being able to access memory on another board, and not being able to process bus-vectored interrupts.

These are guidelines for deciding whether to use serial or parallel bus priority:

- Serial Priority Resolution: It is the easiest to use, requiring no modifications to the boards or the backplane. You can have up to three bus master boards in your system.
- Parallel Priority Resolution: This allows up to 16 master boards. Parallel priority resolution is more complex to set up, and requires special backplane hardware.

Bus Electrical Noise

Signal-to-signal coupling and other noise-inducing phenomena are usually not a problem with Multibus-based systems. But if your system is failing for reasons you can't identify, here are some precautions you can use.

- Place BCLK/ and CCLK/ generators and receivers as close as possible to the Multibus terminators.
- Place any board receiving a signal as close as possible to the board generating the signal.
- If you have empty slots in the backplane, install the boards near the Multibus terminator, leaving the empty slots toward the unterminated end of the backplane. Avoid leaving empty slots between boards.

Refer to the Intel publication USER'S GUIDE TO CONFIGURING MULTIBUS-BASED SYSTEMS for more detailed explanations of these Multibus considerations.

MODIFYING THE iSBC® 215 WINCHESTER DISK CONTROLLER

Regardless which processor board you are using in your system, the iSBC 215 Controller is jumpered the same way. The jumpers are shown in Table 5-3.

Table 5-3. iSBC® 215 Jumpers

Remove Jumper	Add Jumper	Function/Description
W19, C-5	W19, C-4	Interrupt Level 4
	W18, 1-2	16-bit bus compatibility
	W20, 1-3	16-bit bus compatibility
	W21, 1-3	16-bit bus compatibility
	W11, 1-3	If iSBC 218 board is used

Notes: Switches 1 and 2 should be set as follows:
Switch 1 Segment 8 ON, all other segments OFF
Switch 2 Segments 1 and 2 ON, all others OFF

You will have to perform "signal scrambling" for the type of drive you use. Refer to the hardware reference manual for instructions.

The iSBC 215 Controller is documented in the iSBX 215 GENERIC WINCHESTER DISK CONTROLLER HARDWARE REFERENCE MANUAL.

MODIFYING THE iSBX™ 218 DISK CONTROLLER

If you want to add the iSBX 218 multimodule to your iSBC 215 board, you must plug the iSBX 218 multimodule into socket J4 of the iSBC 215 board and modify jumpers on the iSBC 218 board as described in Table 5-4. For jumpering 5 1/4-inch drives, refer to the individual hardware manual.

Table 5-4. Jumpering for the iSBX™ 218 Multimodule™

Add Jumper	Function/Description
W1, A-B	direct memory access
W3, A-C	8-inch disk drives
W4, A-C	" " "
W5, A-C	" " "
W6, A-C	" " "
W7, A-C	" " "

The controller is documented in the iSBX 218 FLEXIBLE DISK CONTROLLER HARDWARE REFERENCE MANUAL.

MODIFYING THE iSBC® 208 FLEXIBLE DISK 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 5-5.

Table 5-5. iSBC® 208 Jumpers

Remove Jumper	Add Jumper	Function/Description
	E79-E84	Interrupt level 5
E45-E49	E41-E45	16-bit I/O address decoding.
E48-E52	E44-E48	Set I/O port address to 180h
E61-E69	E54-E61	
E77-E78		Only if parallel bus priority resolution is used.

The controller is documented in the iSBX 208 FLEXIBLE DISK DRIVE CONTROLLER HARDWARE REFERENCE MANUAL.

MODIFYING THE iSBC® 534 FOUR CHANNEL COMMUNICATIONS EXPANSION BOARD

You must install jumpers on your iSBC 534 Expansion Board to establish the I/O base address and the interrupt level. These jumpers are shown in Table 5-6.

Table 5-6. iSBC® 534 Interrupt and Base Address Jumpers

Remove Jumper	Add Jumper	Function/Description
131-140 132-140	131-138 132-138	Interrupt level 3
	123-126	Base Port Address: 30

Each serial port that you use on the Expansion Board must match the characteristics of the device connected to that port. Although many variations are possible, the most common way of using a port is to connect it to a terminal and to use RS232C protocol. You must prepare one DIP header assembly for each port that you use. Table 5-7 shows how to wire the pins on an 18-pin DIP header assembly for the most common RS232C hookup.

Table 5-7. iSBC® 534 DIP Header Jumpers for RS232C Protocol

Jumper	Description, (with RS232C signals)
4 to 5	Board DSR to Board DTR
6 to 7	Board RTS to board CTS
8 to 10	Board RXD to terminal TXD
9 to 11	Board TXD to terminal RXD
12 to 13	Terminal RTS to terminal CTS
14 to 15	Terminal DSR to terminal DTR

Signal Names:		
	TXD: Transmit Data	RXD: Receive Data
	DTR: Data Terminal Ready	DSR: Data Set Ready
	RTS: Request to Send	CTS: Clear to Send

For any other configuration (current loop, modem hookup, etc.) refer to the iSBC 534 FOUR CHANNEL COMMUNICATIONS EXPANSION BOARD HARDWARE REFERENCE MANUAL.

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 5-8 summarizes jumpers that establish interrupt levels.

Table 5-8. 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)
E70-E73	Level 3, iSBC 534 controller	
E69-E77	Level 4, iSBC 215/218 controller	
E68-E76	Level 5, iSBC 208 controller	(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.
(2) Factory-installed jumpers.

Additional Jumpers

Table 5-9 summarizes additional jumpering of the iSBC 86/12A.

Table 5-9. Other iSBC® 86/12A Jumpers

Remove Jumper	Add Jumper	Function/Description
	E51-E52	Clear To Send signal capability
E125-E126	E127-E128	Set dual-port RAM address
	E12-E21	Only if iSBC 337 is not used
E97-E98	E97-E99	Required by Monitor EPROMs
	E5-E6	Unpopulated memory or port time-out
E151-E152		Only for parallel priority resolution

Parallel Port

Table 5-10 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 5-10. 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 5-11 Shows the settings for each position (segment) of Switch 1.

Table 5-11. 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.

Devices

Table 5-12 describes the devices that must be installed on your iSBC 86/12A Board.

Table 5-12. iSBC® 86/12A Devices

Device	Part Number	Socket
2732A EPROM	145374-001	A28
2732A EPROM	145375-001	A29
2732A EPROM	145376-001	A46
2732A EPROM	145377-001	A47
iSBC 902 Resistor packs	4500645-01	A10, A12, A13
7438 IC	100908-001	A11 (1)
Status Adapter	1002129	A11 (2)

Notes: (1) If parallel port is used for line printer.
(2) If parallel port is used for iSBC 957B package.

The iSBC 86/12A is documented in the iSBC 86/12A SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL.

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 5-13 summarizes jumpers that establish interrupt level jumpers, Table 5-14 summarizes parallel port jumpers, and Table 5-15 shows all other jumpers that must be installed on the iSBC 86/14 Board. Asterisks (*) note indicate factory-installed jumpers.

Table 5-13. Interrupt Jumpers for iSBC® 86/14

Remove Jumper	Add Jumper	Function/Description
E144-E145	E165-E166	Interrupt Level 0, iSBC 337
	E145-E149	Level 1, Non-Maskable Interrupt
	E132-E164	Level 1, Line Printer
	E147-E158	Level 2, System Clock *
	E136-E159	Level 3, If iSBC 534 board used
	E157-E162	Level 5, If iSBC 215/218 board used
	E151-E152	Level 5, iSBC 208 *
	E153-E155	Level 6, Terminal Driver (Read)
	E134-E154	Level 7, Terminal Driver (Write)

Table 5-14. 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

Table 5-15. Other iSBC® 86/14 Jumpers

Remove Jumper	Add Jumper	Function/Description
	E76-E77	Clear To Send signal capability.
E219-E225		Set dual-port RAM address
	E61-E62	Only if iSBC 337 is not used.
E26-E27		Enable Non-Maskable Interrupt
E33-E34		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 is used
	E230-E231	If RAM expansion module not 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 5-16 describes the devices that must be installed on your iSBC 86/14 Board.

Table 5-16. iSBC® 86/14 On-Board Devices

Device	Part Number	Socket
2732A EPROM	145374-001	U57
2732A EPROM	145375-001	U58
2732A EPROM	145376-001	U39
2732A EPROM	145377-001	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.

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 5-17 summarizes jumpers that establish interrupt level jumpers, Table 5-18 summarizes parallel port jumpers, and Table 5-19 summarizes all other jumpers that must be installed on your iSBC 86/30 Board. Asterisks (*) mark factory-installed jumpers.

Table 5-17. Interrupt Jumpers for iSBC® 86/30

Remove Jumper	Add Jumper	Function/Description
E144-E145	E165-E166	Interrupt Level 0, iSBC 337
	E145-E149	Level 1, Non-Maskable Interrupt
	E132-E164	Level 1, Line Printer
	E147-E158	Level 2, System Clock *
	E136-E159	Level 3, iSBC 534 Controller
	E157-E162	Level 5, iSBC 215/218 Controller
	E151-E152	Level 5, iSBC 208 *
	E153-E155	Level 6, Terminal Driver (Read)
	E134-E154	Level 7, Terminal Driver (Write)

Table 5-18. 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	E44-E59	E44-E53	E60-E63
E45-E54	E45-E50	E51-E60	
E46-E55	E45-E54		
E48-E57	E46-E51		
E50-E59	E48-E53		
E51-E60	E50-E52		
E52-E61			

Table 5-19. Other iSBC® 86/30 Jumpers

Remove Jumper	Add Jumper	Function/Description
	E76-E77	Clear To Send signal capability.
E219-E225		Set dual-port RAM address
	E61-E62	Only if iSBC 337 is not used.
E26-E27		Enable Non-Maskable Interrupt
E33-E34		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 is used
	E232-E233	If RAM expansion module not used
E210-E211		For parallel priority resolution
Note: (1) 5 MHz required if iSBC 337 NDP is used.		

Devices

Table 5-20 describes the devices to be installed on an iSBC 86/30 board.

Table 5-20. iSBC® 86/30 On-Board Devices

Device	Part Number	Socket
2732A EPROM	145374-001	U57
2732A EPROM	145375-001	U58
2732A EPROM	145376-001	U39
2732A EPROM	145377-001	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.

CONVENIENCE CHARTS

The next 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 534 Communications Expansion Board
- iSBC 215 Generic Winchester Controller Board
- iSBX 218 Flexible Disk Controller
- iSBC 208 Flexible Disk Controller

The iSBC 208 page also includes Centronics and parallel port signal definitions.

Table 5-21. iSBC® 86/12A Jumpers (Condensed)

Remove Jumper	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 (factory-installed jumper)
	E68-E76	Level 5, iSBC 208 (factory-installed jumper)
	E75-E82	Level 6, Terminal Driver (Read)
	E74-E90	Level 7, Terminal Driver (Write)
	E51-E52	Clear-To-Send signal capability.
E125-E126	E127-E128	Set dual-port RAM address
E97-E98	E97-E99	Required for Monitor in EPROMs
	E12-E21	Only if iSBC 337 is not used.
	E5-E6	Unpopulated memory or port time-out
E151-E152		Only for parallel priority resolution
E13-E14	E13-E27	If parallel port is used for 957B
E19-E20	E14-E30	
E21-E25	E18-E31	
E26-E27	E20-E33	
E30-E31	E25-E31	
E32-E33		
E13-E14	E22-E32	If parallel port is used for line printer
E32-E33		

(1) Applies only to iSBC 86/12A with PWA of 142977-XXX

Table 5-22. iSBC® 86/12A Devices (Condensed)

Device	Part Number	Socket
2732A EPROM	145374-001	A28
2732A EPROM	145375-001	A29
2732A EPROM	145376-001	A46
2732A EPROM	145377-001	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 5-23. iSBC® 86/12A Switch 1 (Condensed)

Segment	Setting	Segment	Setting
1	ON	5	OFF
2	OFF (*)	6	OFF
3	OFF	7	ON
4	OFF	8	OFF

* If iSBC 300 RAM expansion module is not used, this position ON

Table 5-24. iSBC® 86/14 Jumpers (Condensed)

Remove Jumper	Add Jumper	Function/ Description
	E165-E166	Interrupt Level 0, iSBC 337
E144-E145	E145-E149	Level 1, Non-Maskable Interrupt
	E132-E164	Level 1, Line Printer
	E147-E158	Level 2, System Clock (factory-installed jumper)
	E151-E152	Level 5, iSBC 208 (factory-installed jumper)
	E153-E155	Level 6, Terminal Driver (Read)
	E134-E154	Level 7, Terminal Driver (Write)
	E76-E77	Clear-To-Send signal capability
E219-E225		Set dual-port RAM address
E26-E27		Enable Non-Maskable Interrupt
E33-E34		Non-bus vector interrupt
	E61-E62	If iSBC 337 is not used.
	E36-E37	Selects 5MHz clock (required if iSBC 337 is used)
	E124-E125	Selects 2732-type EPROM
E111-E112	E112-E113	EPROM address range
E210-E211		Only for parallel priority resolution
	E119-E120 E232-E233	86/14 with iSBC 300A RAM expansion module
	E230-E231 E232-E233	86/14 without 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 5-25. iSBC® 86/14 Devices (Condensed)

Device	Part Number	Socket
2732A EPROM	145374-001	U57
2732A EPROM	145375-001	U58
2732A EPROM	145376-001	U39
2732A EPROM	145377-001	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

Table 5-26. iSBC® 86/30 Jumpers (Condensed)

Remove Jumper	Add Jumper	Function/ Description
	E165-E166	Interrupt Level 0, iSBC 337
E144-E145	E145-E149	Level 1, Non-Maskable Interrupt
	E132-E164	Level 1, Line Printer
	E147-E158	Level 2, System Clock (factory-installed jumper)
	E151-E152	Level 5, iSBC 208 (factory-installed jumper)
	E153-E155	Level 6, Terminal Driver (Read)
	E134-E154	Level 7, Terminal Driver (Write)
	E76-E77	Clear-To-Send signal capability
E219-E225		Set dual-port RAM address
E26-E27		Enable Non-Maskable Interrupt
E33-E34		Non-bus vector interrupt
	E61-E62	If iSBC 337 is not used.
	E36-E37	Selects 5MHz clock (required with iSBC 337)
	E124-E125	Selects 2732-type EPROM
E111-E112	E112-E113	EPROM address range
E210-E211		Only for parallel priority resolution
	E119-E120	86/30 with iSBC 304 RAM expansion module
	E232-E233	86/30 without RAM expansion module
E44-E53	E44-E59	If parallel port is used for 957B
E45-E54	E45-E50	
E46-E55	E45-E54	
E48-E57	E46-E51	
E50-E59	E48-E53	
E51-E60	E50-E52	
E52-E61		
E44-E53	E60-E63	If parallel port is used for line printer
E51-E60		

Table 5-27. iSBC® 86/30 Devices (Condensed)

Device	Part Number	Socket
2732A EPROM	145374-001	U57
2732A EPROM	145375-001	U58
2732A EPROM	145376-001	U39
2732A EPROM	145377-001	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

Table 5-28. iSBC® 215 Jumpers (Condensed)

Remove Jumper	Add Jumper	Function/Description
W19, C-5	W19, C-4	Interrupt Level 4
	W18, 1-2	16-bit bus compatibility
	W20, 1-3	16-bit bus compatibility
	W21, 1-3	16-bit bus compatibility
	W11, 1-3	If iSBC 218 board is used

Notes: Switches 1 and 2 should be set as follows:
Switch 1 Segment 8 ON, all other segments OFF
Switch 2 Segments 1 and 2 ON, all others OFF

You will have to perform "signal scrambling" for the type of drive you use. Refer to the hardware reference manual for instructions.

iSBX™ 218

Table 5-29. Jumpering for the iSBX™218 Multimodule™(Condensed)

Add Jumper	Function/Description
W1, A-B	direct memory access
W3, A-C	8-inch disk drives
W4, A-C	" " "
W5, A-C	" " "
W6, A-C	" " "
W7, A-C	" " "

Table 5-30. iSBC® 534 Jumpers (Condensed)

Remove Jumper	Add Jumper	Function/Description
131-140 132-140	131-138 132-138	Interrupt level 3
	123-126	Base Port Address: 30

Table 5-31. iSBC® 534 DIP Header Jumpers, RS232C (Condensed)

Jumper	Description, (RS232C Signal Names)
4 to 5	Board DSR to Board DTR
6 to 7	Board RTS to board CTS
8 to 10	Board RXD to terminal TXD
9 to 11	Board TXD to terminal RXD
12 to 13	Terminal RTS to terminal CTS
14 to 15	Terminal DSR to terminal DTR
Signal Names:	TXD: Transmit Data RXD: Receive Data DTR: Data Terminal Ready DSR: Data Set Ready RTS: Request to Send CTS: Clear to Send

Table 5-32. 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 installed, for serial bus priority resolution).

Line Printer

Table 5-33. 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

GROUNDS:

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

***

CHAPTER 6. SYSTEM MANAGEMENT

This chapter is addressed to the system manager -- the person who is responsible for managing an iRMX 86 PC System. To the Operating System, the system manager is simply anyone with user ID 0 (zero). As the system is delivered from Intel, however, user ID 0 is not automatically assigned to a terminal. To assume user ID 0, an operator uses the SUPER command (see Chapter 3) and must know the password associated with SUPER. When Intel sends you the System Diskette, the SUPER password is null, so you reply to the prompt for a password by typing a carriage return.

Beginning here, this chapter assumes that you are the system manager in both senses of that term, that is, you are responsible for managing your system, and you operate with user ID 0.

The important considerations about your installation of the iRMX 86 PC package are the following:

- Will your system support multiple terminals?
- Will more than one user be using the system?
- Do you wish to protect files and devices from unauthorized access? "Unauthorized access" means both deliberate attempts to access files and accidental access. Naive or careless users, as well as hostile users, should be prevented from deleting or corrupting files on a system used by more than one person. The iRMX 86 PC system contains software mechanisms to make possible a protected environment. Even if your system has only one keyboard terminal, but more than one person uses it, protection features can prevent unauthorized access.

This chapter contains these major sections:

- How to copy the System Diskette.
- iRMX 86 PC System Diskette: The contents of the diskette that Intel sends you. The term System Diskette designates the diskette that Intel delivers as part of the iRMX 86 PC product, or a copy of this diskette. (The entire package is described in the Preface.)
- Editing Terminal and User Definition Files: How to change the files and directories to meet the requirements of your individual system.
- Other System Management Functions: How to attach hardware devices and close down a multi-user system.

NOTE

After a system is bootstrap loaded, the logical name (prefix) :SD: designates the disk from which the system was booted.

COPYING THE iRMX™ 86 PC SYSTEM DISKETTE

When you receive the iRMX 86 PC product, and have prepared the hardware on which it will run, you should bootstrap load the system (described in the beginning of Chapter 1) and backup the System Diskette. Then store the Intel-supplied System Diskette in a safe place.

On the System Diskette is a file, :SD:BACKUPSYS that will automatically create a new system disk with the correct directories and files. To use this file, you:

1. Type SUPER and respond to the password prompt with a RETURN key.
2. Attach the disk drive to which you will copy the disk. Attach it as :F1:.
3. Format the disk.
4. SUBMIT the file BACKUPSYS, which contains all other commands necessary to complete the backup process. (If you are not familiar with the SUBMIT command, it is described in Chapter 3.)

Here is a detailed description of these steps:

1. Assume user ID 0. Type SUPER, and reply to the password prompt with a carriage return (RETURN key). The System Diskette is delivered without a password; we show later how to establish a password.
2. Attach the disk drive. Issue an appropriate ATTACHDEVICE command to attach the disk drive on which you will create the new system diskette. Attach the drive as logical name :F1:. By appropriate, we mean with the correct physical name for the drive. These physical names are listed with the description of the ATTACHDEVICE command in Chapter 3. If you are not familiar with the ATTACHDEVICE command, you should refer to the description.

   Two examples show how to type the ATTACHDEVICE command. Assume that you have bootstrap loaded the Operating System from Drive 0 connected to an iSBX 218. If you intend to copy to the second drive on the same controller, you will issue the following command:

```
ATTACHDEVICE WFD1 AS F1
```

But if you intend to copy to a Priam 3450 Winchester Disk, issue:

```
ATTACHDEVICE IW0 AS F1
```

If the diskette in the second drive has not been formatted, you will receive an error message on the screen saying something like

```
F1    volume not formatted
```

followed by one more line of error information. Ignore the messages; the drive is attached anyway.

3. Format the disk. If you are formatting another flexible diskette, the following command is appropriate:

```
FORMAT :F1:NEWDSK  FILES = 100 INTERLEAVE = 7
```

If you are formatting a Winchester disk, the following command is generally appropriate:

```
FORMAT :F1:newsys FILES = 5000  INTERLEAVE = 4.
```

Note that the FORMAT command for the flexible disk specifies an interleave value of 7. This is because if you did not specify a value, the FORMAT command uses a default value of 5. With an interleave factor of 5, booting the system from a flexible diskette can take nearly three minutes, instead of about one-half minute for a diskette formatted with an interleave factor of 7.

You may wish to adjust the FILES parameter if you have unusual file system requirements. But the values shown are proper for most installations.

4. SUBMIT the file :SD:BACKUPSYS. It creates the correct directories on the new device, copies all of the proper files, and establishes the owner and access rights for each.

Figure 6-1 shows the commands in BACKUPSYS.

```
COPY :SD:GSYS.020 TO :F1:GSYS.020
CREATEDIR :F1:SYSTEM
CREATEDIR :F1:CONFIG
CREATEDIR :F1:CONFIG/USER
CREATEDIR :F1:USER
CREATEDIR :F1:WORK
CREATEDIR :F1:LANG
CREATEDIR :F1:UTILS
COPY :SYSTEM:* TO :F1:SYSTEM/*
COPY :SD:CONFIG/TERMINALS TO :F1:CONFIG/TERMINALS
COPY :SD:CONFIG/USER/* TO :F1:CONFIG/USER/*
PERMIT :F1:USER L USER= WORLD
PERMIT :F1:SYSTEM L USER = WORLD
PERMIT :F1:UTILS L USER= WORLD
PERMIT :F1:LANG L USER= WORLD
PERMIT :F1:WORK DLAC USER= WORLD
PERMIT :F1:SYSTEM/* R USER=WORLD
PERMIT :F1:SYSTEM/RMX86 N USER=WORLD
PERMIT :F1: L USER = WORLD
COPY :SD:BACKUPSYS TO :F1:BACKUPSYS
PERMIT :F1:BACKUPSYS R USER = WORLD
PERMIT :F1:GSYS.020 R USER = WORLD
CREATEDIR :F1:USER/0
CHANGEID 1
CREATEDIR :F1:USER/1
CHANGEID 2
CREATEDIR :F1:USER/2
CHANGEID 3
CREATEDIR :F1:USER/3
CHANGEID 4
CREATEDIR :F1:USER/4
CHANGEID 65535
CREATEDIR :F1:USER/65535
```

Figure 6-1. :SD:BACKUPSYS

When you have an editor installed on your system, you may wish to change the password on the copy of the System Diskette. This will prevent unauthorized users from using the the copy to gain access to files that you want protected. We do not recommend creating a password on the System Diskette you receive from Intel; keeping it in a secure place is safer than modifying the diskette.

The password used with SUPER is in the user definition file for user ID 0. This file is described in a later section, EDITING TERMINAL AND USER DEFINITION FILES, which explains how to protect your system from accidental or deliberate access.

iRMX™ 86 PC SYSTEM DISKETTE

This section describes the file structure of the System Diskette, by listing the names of the files, describing the information in these files, and describing the directory structure.

Figure 6-2 shows the directories and data files that exist on the iRMX 86 PC System Diskette. The figure also shows:

- Owner's User ID. The owner is important because that user always has the ability to change the access rights associated with the file. Therefore the owner can always access the file and, if it is a directory, any files at lower levels in the file tree.
- Access Rights of WORLD. WORLD access is important because it indicates the access rights of all other users.

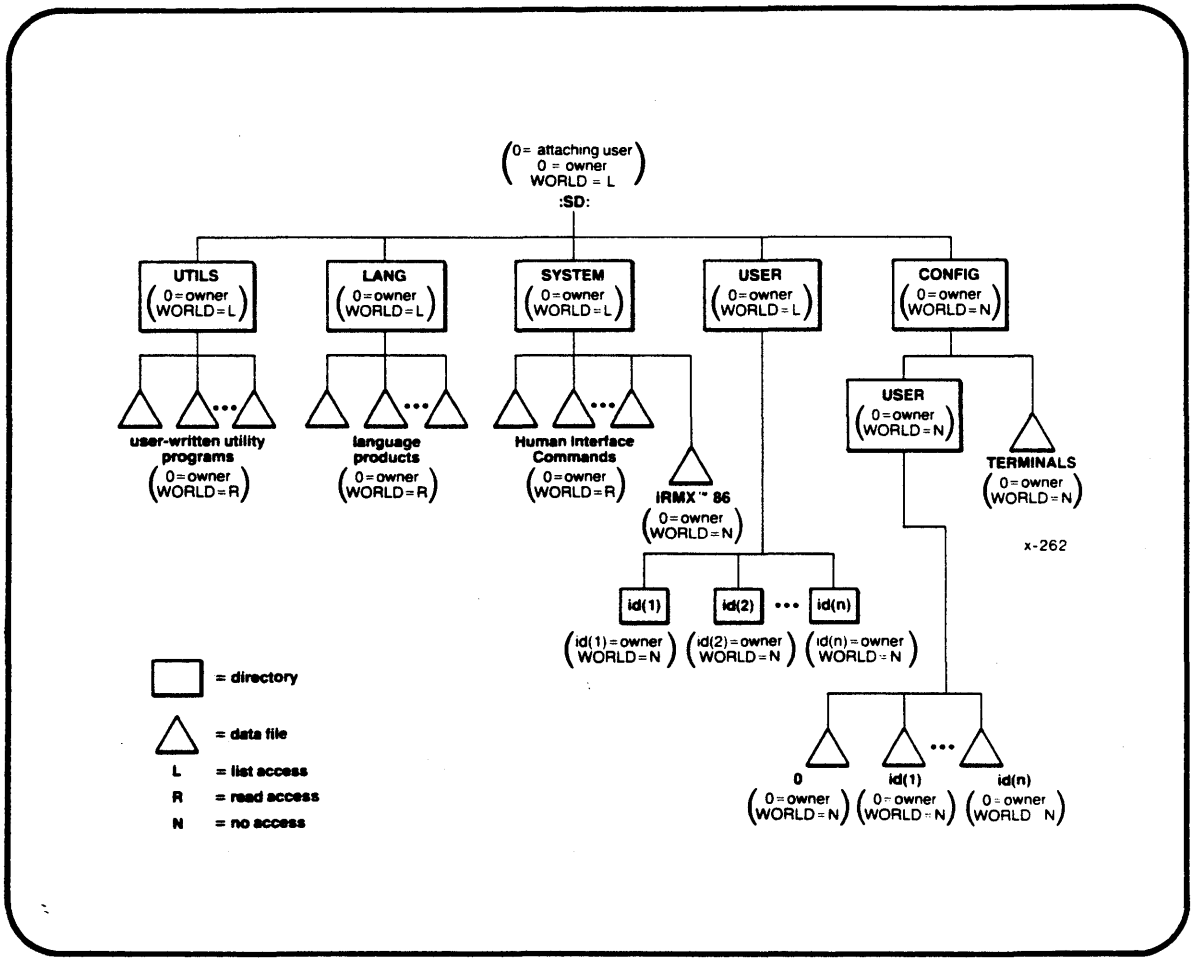


Figure 6-2. File Structure of the System Device

In the figure, :SD: indicates the root directory of the device. You can bootstrap load the system from any disk on your system; the Bootstrap Loader recognizes the device from which it loads and causes that device to be assigned logical name :SD:. You can attach other devices after the system has been booted by a procedure described in a later section.

User ID 0 (the system manager) is the owner of the :SD: device. This prevents users other than the system manager from detaching the system device. Only user 0 has all access rights to the root directory. Other users have List access to allow them to view the files in the root directory. However, they do not have change entry access to the root directory, nor do they have delete access or add entry access.

Three first-level directories listed in Figure 6-2 are used for commands, for language processors such as compilers, and for utilities. They are SYSTEM, LANG, and UTILS respectively. User ID 0 should be the owner of these directories and the files they contain. To protect language processors and utilities, other users should have List access to the directories (to be able to see what is available) and Read access to the command files (to be able to run the commands).

The directory WORK is used by language utilities for creating temporary files.

User ID 0 is the owner of the bootstrap-loadable iRMX 86 PC Operating System (the file RMX86 in the SYSTEM directory), and other users have no access to it. This prevents users from creating their own system volumes.

The first-level directory USER contains the directories which are the default prefixes of the users. User ID 0 is the owner of USER and other users have only List access. However, the directories contained in USER are owned by the corresponding user IDs with no access to other users. By owning their default prefix directory, a user can change the access rights. This allows one to permit other users to access files in the default directory. However, since no other users have automatic access to the default directory, the user can maintain privacy if desired.

The figure only generally shows the user directories. The System Diskette contains user directories for IDs 0 through 4, and 65535 (WORLD).

The other first-level directory (CONFIG) contains the terminal and user definition files, also called configuration files. User ID 0 is the owner of the directory and the files contained in it. Other users have no access to the directory or to the files it contains. This prevents everyone but the system manager from modifying the terminal and user configuration. It also prevents users from discovering the system manager's password. The CONFIG directory and the files described here must reside on the :SD: device.

Two files not shown in the figure are in the root directory: BACKUPSYS, and GSYS.020. BACKUPSYS is described earlier in the chapter. GSYS.020 is the identifier file for the iRMX 86 PC Operating System. An example command in Chapter 2 shows its contents.

EDITING THE TERMINAL AND USER DEFINITION FILES

To control file access and terminals assignments on a system, you edit a few files to make changes and then re-boot the system. This discussion assumes that you have copied your System Diskette, and have installed an editor on your system.

If only one person accesses your system, that user requires access to all files and all devices in the system, and there is no need to restrict access to files or devices.

However, with more than one person using the system, file and device access becomes an issue. Some reasons for this are the following:

- To maintain system security, you should limit access to the Configuration Files.
- Some users might want to prohibit other users from reading their files or viewing their directories. However, some users might want to grant other users the ability to access their files. To do this, users should be the owners of the directories that serve as their default prefixes.
- All users should be able to run Human Interface commands, and be able to use software packages that you install on your system (such as compilers or editors). To do so, they require the ability to read the files containing the commands and utilities. However, to protect the files from damage, you should restrict other types of access to these files.
- Some devices (such as a hard disk or the device that serves as the system device) should be available to all users. However, to protect users who access these devices, only the system manager should be able to detach the devices.

To create a multi-access system that protects files and directories, someone (normally the system manager) must set up the correct file structure before allowing other users to access the system. The following sections describe this process.

In this section we describe in detail:

- The general format of the terminal definition file and of the user definition files.
- The contents of these files on the System Diskette you receive from Intel.
- How to change parameters to make changes in the operation of your system. For example, by editing these files you can change the amount of memory allocated to a particular terminal user; set the SUPER password, and you can specify which terminals are used.

The two kinds of files that the system manager edits to control users and terminals are: the terminal definition file, and user definition files (one for each user). The pathnames of the files are:

file	pathname
Terminal definition file	CONFIG/TERMINALS
User definition files	CONFIG/USER/id

Where id is the User ID of the corresponding user. The iRMX 86 PC System Diskette contains one user definition file for each of the following User IDs:

0 (system manager)
1
2
3
4
65535 (WORLD)

TERMINAL DEFINITION FILES

The terminal definition file (:SD:CONFIG/TERMINALS) defines all terminals through which users intend to access the Human Interface. The file consists of several lines of information which can be divided into two parts:

1. One line consisting of an integer, indicating the number of terminals to be connected. This is the first line of the file.

2. Device name and attributes of the terminals, one terminal per line.

The following is a list of the contents of the terminal definition file on the iRMX 86 PC System Diskette:

```
1
T0,65535,64,200
T1,1,225,200
T2,2,64,200
T3,3,64,200
T4,4,64,200
```

The first line in the file, 1, defines how many terminals are active, and therefore how many interactive jobs to create when the system is initialized. Each succeeding line defines one terminal. You receive the System Diskette with only the first terminal (T0) activated. If you change this number to 3, you activate T0, T1, and T2. Changing the number to 5 activates all the terminals that the iRMX 86 PC System can support.

The device name and attributes of a terminal must reside on a single line, with commas separating the individual elements. Only the first two elements are required. Embedded blanks are not allowed. For example, the terminal connected to the Port 0 on an iSBC 534 controller board is defined by the third line of the file:

```
T1,1,200,200
```

The following general description will help you can edit the file to make it fit your system. A terminal definition line consists of the following parameters:

```
device-name,user-id,partition-size,max-priority,init-pathname
```

where:

device-name — Name of the terminal; in the example: T1. The names were established when the iRMX 86 PC Operating System was configured. These names, and corresponding hardware ports, are:

T0	Serial port on iSBC 86 Single Board Computer
T1	Serial Port 0, iSBC 534 Controller
T2	Serial Port 1, iSBC 534 Controller
T3	Serial Port 2, iSBC 534 Controller
T4	Serial Port 3, iSBC 534 Controller

user-id — Decimal number in the range 0 through 65535 that represents the ID of the user associated with this terminal; in the example: 1. You can assign the same User ID to more than one terminal. For any value you specify in this field, there must be a corresponding user definition file. (The next section describes user definition files.)

partition-size — Size of the memory partition assigned to this user, specified in 1024-byte (1K) units; in the example: 64. The memory partition is an area of memory in which the user can load and run programs. In order to run compilers, you will need 225K bytes. As we send you the system, the user assigned to the terminal on the single board computer is allocated 64K bytes. This is to ensure that you will be able to boot the the system with the minimum memory required on the system. This field for Terminal 1 is given 225K bytes, but is not active when you boot the system.

max-priority — Decimal number specifying the maximum priority that any tasks associated with this user can have; in the example: 200.

init-pathname — Pathname of the file containing the user's initial program; this parameter is not in the example. This is the program that begins running when the Human Interface creates the interactive job for the user.

Omitting Unnecessary Parameters

Remember that only device-name and user-id are required. The rules for omitting values are:

- If you omit one of the intermediate optional parameters but specify a later one, you must include a comma as a place holder for the parameter. For example, if for T1 you specify an initial program pathname, but do not want to specify partition-size or max-priority, the line might look like:

  ```
  T1,1,,,MYFILE
  ```

- If you omit a value, the Human Interface uses the value specified in the user definition file for this User ID. (The next section describes the user definition file.) If the terminal definition file and a user definition file conflict, the Human Interface uses the value specified in the the terminal configuration file.

Order of Terminal Definition Lines

When the Human Interface starts running, it creates jobs for the terminals in the order they are specified in the terminal definition file. You should define the terminals in order of their importance to guarantee that the most important terminals have access to the system. If there is no order of importance, you should specify the terminals in order of their partition sizes, with largest partition sizes first.

User Definition Files

User definition files define the attributes of Human Interface users. There must be a separate file for each user. Each file contains information (called attributes) for the user. The attributes can be separated by commas or appear on separate lines. When separated by commas, embedded blanks are not allowed.

For example, the contents of the user definition file for User ID 1 on the iRMX 86 PC System Diskette (:SD:CONFIG/USER/1) contains the following data: (the description refers to this example).

```
1,,64,128,190,:SD:USER/1
```

The general format of each user definition file is as follows:

```
user-id
password
default-partition-size
maximum-partition-size
max-priority
default-prefix-pathname
init-pathname
```

where:

user-id	Decimal number in the range 0 through 65535 that represents the ID of the user; in the example, 1.
password	A one- to eight-character password that is associated with this user. Currently, this field applies only to User ID 0; and is the password that a user must enter to use the SUPER command. For user IDs other than 0, this field is reserved for future use; enter a null value (comma or carriage return used as a place holder), as shown in the example.
default-partition-size	Size, in 1024-byte units, of the memory partition that the Human Interface assigns to the user's interactive job. This value is used unless overridden by the value in the terminal definition file. In the example, the value is 64.
maximum-partition-size	Size, in 1024-byte units, of the largest memory partition that the Human Interface can assign to the user's interactive job. The suggested value for users who run only standard Human Interface commands is 32. Users who run language products (such as compilers or linkers) require a larger value. In the example, the size is 128.
max-priority	Number specifying the maximum priority that any tasks associated with this user can have. This value is used unless overridden by the value in the terminal definition file. The value is in the example is 190.
default-prefix-pathname	Pathname of the directory that serves as this user's default prefix (corresponding to the :HOME: and initial :$: directory). The directory specified in this field must exist or the user will be unable to access the Human Interface. In the example, the default prefix directory is :SD:USER/1.
init-pathname	Pathname of the file containing the user's initial program. This is the program that begins running when the Human Interface creates the interactive job for the user. If you omit this value, the Human Interface uses its standard command line interpreter (CLI) as the initial program. It is omitted for all users on the iRMX 86 PC System Diskette.

Following is a list of the contents of each user definition file that you receive with the iRMX 86 PC System Diskette.

User ID	File	Contents
0	:SD:CONFIG/USER/0	0,,64,128,190,:SD:USER/0
1	:SD:CONFIG/USER/1	1,,64,128,190,:SD:USER/1
2	:SD:CONFIG/USER/2	2,,64,128,190,:SD:USER/2
3	:SD:CONFIG/USER/3	3,,64,128,190,:SD:USER/3
4	:SD:CONFIG/USER/4	4,,64,128,190,:SD:USER/4
WORLD	:SD:CONFIG/USER/65535	65535,,64,128,190,:SD:USER/65535

OTHER SYSTEM MANAGEMENT FUNCTIONS

You should consi'er at least two other system manager functions: attaching hardware devices and shutting down the system in an orderly fashion.

ATTACHING HARDWARE DEVICES

After the system is initialized you can add any new devices to the system with the ATTACHDEVICE command. If you have many devices that must be attached to the system, you may wish to attach them with a submit file. This could be the logon file for the User ID of the system terminal (the terminal connected to the serial line on the Single Board Computer . The iRMX 86 PC System Diskette assigns WORLD to the system terminal, which means that if you include the ATTACHDEVICE commands in the logon file for this user (:SD:USER/65535/PROG/R?LOGON), any user can detach the device. If you want to restrict who can detach devices, you can build a separate SUBMIT file which only you can access. You then manually issue SUPER to become User ID 0, and then SUBMIT the file.

Another handy arrangement is to assign each user a flexible disk drive. When the user starts a terminal session, the logon file can attach the device.

SHUTTING DOWN THE SYSTEM

You can methodically shut down a multi-user system by the following process:

1. Issue INITSTATUS to get the identifying number of each job.
2. Issue LOCK.
3. Use JOBDELETE to delete each job.
4. Use DETACHDEVICE with FORCE to detach the devices.

***

CHAPTER 7. 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 Release 5 iRMX 86 SYSTEM is a self-contained summary of information you need to use the iRMX 86 PC Operating System. Much of the information in this manual is repeated in some form 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

  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

  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

  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.

- iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL

  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 OPERATOR'S MANUAL

  This manual describes the iRMX 86 commands -- the same commands described in Chapter 3 of this manual. In addition, the manual describes how to use the Files Utility and the Patch Utility.

- iRMX 86 HUMAN INTERFACE REFERENCE MANUAL

  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 LOADER REFERENCE MANUAL

  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 DISK VERIFICATION UTILITY REFERENCE MANUAL

  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 PROGRAMMING TECHNIQUES MANUAL

  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

  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

  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

  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.

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

  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

  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.

- ASM86 LANGUAGE REFERENCE MANUAL

  This manual documents the 8086/8087/8088 macro assembly language, ASM86. It describes the assembly language instructions and the macro processing language.

- ASM86 MACRO ASSEMBLER OPERATING INSTRUCTIONS FOR 8086-BASED DEVELOPMENT SYSTEMS

  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

  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.

- iAPX 86,88 FAMILY UTILITIES USER'S GUIDE

  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

  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

  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.

- USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE

  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

- iSBC 86/12A SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL and

- iSBC 86/14 and iSBC 86/30 SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL

  These two manuals describe, for each computer, principles of operation and how to incorporate iSBC Multimodule units (like on-board RAM and and the 8087 Numeric Processor Extension).

DISK CONTROLLERS

- iSBX 208 FLEXIBLE DISK CONTROLLER HARDWARE REFERENCE MANUAL
- iSBC 215 GENERIC WINCHESTER DISK CONTROLLER HARDWARE REFERENCE MANUAL
- iSBX 218 FLEXIBLE DISK CONTROLLER HARDWARE REFERENCE MANUAL

  These manuals describes specifications, jumper configurations, programming considerations, and principles of operation of the respective Disk Controllers.

COMMUNICATION EXPANSION BOARD

- iSBC 534 FOUR CHANNEL COMMUNICATIONS EXPANSION BOARD HARDWARE REFERENCE MANUAL

MEMORY BOARDS

- iSBC 016A/032A/064A/028A/056A RAM MEMORY BOARD HARDWARE REFERENCE MANUAL
- iSBC 016A/032A/064A/028A/056A RAM BOARDS HARDWARE REFERENCE MANUAL

  These manuals describe specifications, jumper configurations, programming considerations, and principles of operation of the respective memory boards.

CHASSIS/POWER SUPPLY

- iSBC 680/681 MULTISTORE USER SYSTEM PACKAGE HARDWARE REFERENCE MANUAL

  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

Table A-2 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.

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	*	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$EXIST	* * * * *	A token parameter has a value which is not the token of an existing object.
7H	E$STATE	*	A task attempted an operation which would have caused an impossible transition of a task's state.
8H	ENOTCON-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.

N Nucleus Reference Manual
B Basic I/O System Ref Manual
E Extended I/O Sys Ref Manual
L Loader Reference Manual
H Human Interface Reference Manual

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
		Environmental Conditions (continued)	
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$-ENTRY	* *	The specified slot in a directory file is empty.
25H	EDIREND	* *	The specified slot is beyond the end of a directory file.
26H	E$FACCESS	* * * *	File access not granted.
27H	E$FTYPE	* * *	Incompatible file type.
28H	E$SHARE	* * * *	Improper file sharing requested.
29H	E$SPACE	* *	No space left.
2AH	E$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	* * *	Invalid volume name.
2EH	EDEVOFF-LINE	*	The device being accessed is now offline.

N Nucleus Reference Manual
B Basic I/O System Ref Manual
E Extended I/O Sys Ref Manual
L Loader Reference Manual
H Human Interface Reference Manual

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
		Environmental Conditions (continued)	
2FH	E$IFDR	* *	Invalid file driver request.
40H	ELOGNAME$-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	ELOGNAME$-NEXIST	* *	The Extended I/O System was unable to find a specified logical name in the object directories that it checks.
46H	ENOTOWNER	*	The user who attempted to detach the device is not the owner of the device.
47H	EIOJOB	*	The Extended I/O System cannot create an I/O job because the size specified for the object directory is too small.
50H	EIOUNCLASS	*	An unknown type of I/O error occurred.
51H	EIOSOFT	* *	A soft I/O error occurred. A retry might be successful.
52H	EIOHARD	* *	A hard I/O error occurred. A retry is probably useless.

N Nucleus Reference Manual
B Basic I/O System Ref Manual
E Extended I/O Sys Ref Manual
L Loader Reference Manual
H Human Interface Reference Manual

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
		Environmental Conditions (continued)	
53H	EIOOPRINT	* *	The device was off-line. Operator intervention is required.
54H	EIOWRPROT	* *	The volume is write-protected.
60H	EABSADD-RESS	*	An absolute object program was loaded into system protected memory area.
61H	EBADGROUP	* *	Invalid group component in the a group definition record.
62H	EBAD-HEADER	* *	Invalid header record in the object file.
63H	EBADSEG-DEF	* *	Invalid 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	* *	Invalid fixup record in the object file.
67H	ENOLOADER $MEM	* *	Insufficient memory to satisfy loader dynamic memory requirements.
68H	ENOMEM	* *	Insufficient memory to create PIC/LTL segments.
69H	ERECFOR-MAT	* *	Invalid record format encountered.
6AH	EREC-LENGTH	* *	Record length of an object record exceeds configured loader-buffer size.

N Nucleus Reference Manual
B Basic I/O System Ref Manual
E Extended I/O Sys Ref Manual
L Loader Reference Manual
H Human Interface Reference Manual

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
		Environmental Conditions (continued)	
6BH	ERECTYPE	* *	Invalid record type encountered in the object file.
6CH	ENOSTART	* *	Start address not found.
6DH	EJOBSIZE	* *	Maximum job-size specified is less than the memory requirement specified in the object file.
6EH	E$OVERLAY	*	Overlay name does not match with any of the overlay module names.
6FH	E$LOADER $SUPPORT	* *	The object file being loaded requires features not supported by the configured loader.
70H	ESEG BOUNDS	*	One of the data records in a module loaded by the Application Loader referred to an address outside the segment created for it.
80H	E$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$SEPARA- TOR	*	The parse buffer contains a command separator.
83H	E$CONTINUED	*	The parse buffer contains a continuation character.
84H	E$INVALID$- NUMERIC	*	A numeric value contains invalid characters.

N Nucleus Reference Manual
B Basic I/O System Ref Manual
E Extended I/O Sys Ref Manual
L Loader Reference Manual
H Human Interface Reference Manual

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
		Environmental Conditions (continued)	
85H	E$LIST	*	The last value of the value list is missing.
86H	E$WILDCARD	*	A wild-card character appears in an invalid context, such as an intermediate component of a pathname.
87H	E$PREPOSI-TION	*	The same preposition as on the the command line was indicated, but can not be used.
88H	E$PATH	*	The command line specifies an invalid pathname.
89H	E$CONTROL$C	*	The user typed CONTROL-C while the command was being loaded.
8AH	E$CONTROL	*	The command line contains an invalid control.
8BH	E$UNMATCHED $LISTS	*	There were no more input pathnames although the output pathname list was not empty.

N Nucleus Reference Manual
B Basic I/O System Ref Manual
E Extended I/O Sys Ref Manual
L Loader Reference Manual
H Human Interface Reference Manual

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
		Programmer Errors	
8000H	E$ZERO$-DIVIDE	*	A task attempted to divide by zero.
8001H	E$OVERFLOW	*	An overflow interrupt occurred.
8002H	E$TYPE	* * * * *	A token parameter referred to an existing object that is not of the required type.
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.
8005H	EBADCALL	* *	The I/O System code has been damaged, probably due to a bug in an application task. Recovery is not possible.
8006H	E$ARRAY$-BOUNDS	*	Hardware or software has detected an array overflow.
8007H	ENDP-STATUS	*	An 8087 Numeric Processor Extension error has been detected; Operating System extensions can return the status of the 8087 to the exception handler.
8008H	E$CHECK$EX-CEPTION	*	A software interrupt 17 has occurred.
8021H	E$NOUSER	* * *	No default user.
8022H	E$NOPREFIX	* * *	No default prefix.
8040H	ENOTLOG$-NAME	* *	Specified object is not a device connection or file connection.

N Nucleus Reference Manual
B Basic I/O System Ref Manual
E Extended I/O Sys Ref Manual
L Loader Reference Manual
H Human Interface Reference Manual

Table A-2. iRMX™ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
	Programmer Errors (continued)		
8041H	ENOT-DEVICE	*	A token parameter referred to an existing object that is not, but should be, a device connection.
8042H	ENOTCON-NECTION	*	A token parameter referred to an existing object that is not, but should be, a file connection.
8060H	EJOBPARAM	* *	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.
8081H	EJOB-TABLES	*	An internal Human Interface table was overwritten, causing it to contain an invalid value.
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
B Basic I/O System Ref Manual
E Extended I/O Sys Ref Manual
L Loader Reference Manual
H Human Interface Reference 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 to 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 Include 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 subsystem of the Operating System.

LAYERS OF THE iRMX™ 86 SYSTEM

The 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 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.

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 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 discipline.
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.
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$MIN	Dynamically changes the minimum memory requirements of the job environment containing the calling task.
SET$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.

These system calls are considered System Programmer calls because of their global effect on the system.

AGETEXTENSION$DATA	Returns from the I/O System extension data stored with a file.
A$PHYSICAL$ATTACH$DEVICE	Attaches a device to the Basic I/O System.
A$PHYSICAL$DETACH$DEVICE	Detaches a device from the Basic I/O System.
ASETEXTENSION$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$USER	Creates a user object.
DELETE$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.
SETOSEXTENSION	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.

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$CONNECTION	Deletes the connection to the specified file.
AGETFILE$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.
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.

CREATEIOJOB	Creates an I/O job with one task.
EXITIOJOB	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$IO$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$EXIT$IO$JOB	Sends a message to a previously designated mailbox and deletes the calling task.
SGETCONNECTION$STATUS	Provides status information about file and device connections.
SGETFILE$STATUS	Allows a task to obtain information about a physical, stream, or named file.
S$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.
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$CONNECTION	Creates a command connection and returns a token.
C$DELETE$COMMAND$CONNECTION	Deletes a specific command connection.
C$FORMAT$EXCEPTION	Formats a default message into a user buffer for a given exception code.
CGETCHAR	Gets a character from the command line.
CGETINPUT$CONNECTION	Returns an EIOS connection for the specified input file.
CGETINPUT$PATHNAME	Parses the command line return a pathname that will identify the Standard Input file.
CGETOUTPUT$CONNECTION	Returns an EIOS connection for the specified output file.
CGETOUTPUT$PATHNAME	Parses the command line and returns a pathname that will identify the Standard Output file.
CGETPARAMETER	Parses the command line for the next parameter and returns it as a keyword name and a value.
C$SEND$CO$RESPONSE	Sends a message to the command output (CO) and reads a response from the command input (CI).
C$SEND$COMMAND	Concatenates command lines into the data structure created by CREATE$COMMAND$CONNECTION and then executes command.

C$SEND$EO$RESPONSE	Sends a message to the error output device (EO) and returns a response from the error input device (EI).
CSETCONTROL$C	Changes calling program's CONTROL C semaphore to the specified semaphore.
CSETPARSE$BUFFER	Parses a buffer other than the current command line.

***

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."

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 dpending 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"
<B>	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 00-0FFH. 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 0000-0FFFFH. 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.

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.

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	$-2x10^{9} < X < +2x10^{9}$
Long integer	H	64	19	$-9x10^{18} < X < +9x10^{18}$
Packed decimal	H	80	18	-99..99 < X < +99..99(18 digits)
Short real*	R	32	6-7	$8.43x10^{-37} < \|X\| < 3.37x10^{38}$
Long real*	R	64	15-16	$4.19x10^{-307} < \|X\| < 1.67x10^{308}$
Temporary real	R	80	19	$3.4x10^{-4929} < \|X\| < 1.2x10^{4929}$

* The short and long real data types correspond to the single and double precision data types defined in other Intel numeric products.

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.

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 dislayed 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  3FFF199999999999999AR  .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  43118B54F22AEB00R  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
```

If the magnitude of a number is greater than or equal to 0.1 and is less than 10**<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 hexadeximal 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 date 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 OFFFFh. When a range is expected and neither an ending address nor a length is specified, the range is taken to be a single byte.

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.

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.

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.

- 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.

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>

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

***

intel®

REQUEST FOR READER'S COMMENTS

ntel Corporation attempts to provide documents that meet the needs of all Intel product users. This form lets ou participate directly in the documentation process.

lease restrict your comments to the usability, accuracy, readability, organization, and completeness of this locument.

. Please specify by page any errors you found in this manual.

. Does the document cover the information you expected or required? Please make suggestions for improvement.

. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

. Did you have any difficulty understanding descriptions or wording? Where?

. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

AME _____ DATE _____

ITLE _____

OMPANY NAME/DEPARTMENT _____

DDRESS _____

ITY _____ STATE _____ ZIP CODE _____

lease 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

OMO Technical Publications