

iRMX 86™ INSTALLATION GUIDE

Order Number: 9803125-04

REV.	REVISION HISTORY	PRINT DATE
-001	Original Issue	4/80
-002	Update to reflect Release 2 of the iRMX 86 software and hardware requirements.	10/80
-003	Update to reflect changes to support Release 3 of the iRMX 86 Operating System; new chapter for the Start-Up System; the Files Utility chapter is moved to Chapter 8; new hardware information to support new controller boards; various minor technical and typographical errors are corrected.	5/81
-004	Update to reflect changes to support Release 4 of the iRMX 86 Operating System; change to the Patching Utility; and new hardware information to support new controller boards.	11/81

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).

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

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

BXP
CREDIT
i
ICE
iCS
im
INSITE
Intel
Intel

Intelevison
Intellec
iRMX
iSBC
iSBX
Library Manager
MCS
Megachassis
Micromainframe

Micromap
Multibus
Multimodule
Plug-A-Bubble
PROMPT
Promware
RMX/80
System 2000
UPI
µScope

and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or RMX and a numerical suffix.

PREFACE

The iRMX 86 Operating System is a software package that provides a real-time, multitasking environment for Intel iAPX 86,88-based single board computers. This manual provides you with the information required to install your first operating system.

NOTE

Although you can configure an iRMX 86 system for use with an iSBC 86/05, 88/25, or 88/40 board, this manual assumes that you will be installing your first system using the iSBC 86/12A board, and much of the installation instruction is based upon that premise. Therefore, not all of the material given in the manual is applicable if you plan to build a custom board from components. In any event, use of the iSBC 86/12A board is suggested for system prototyping and debugging purposes. Portions or all of the information in the following chapters is relevant for component users:

- Chapter 1. Introduction
- Chapter 2. iRMX 86 Development Environment
- Chapter 5. Start-Up System
- Chapter 6. Patching Utility
- Chapter 7. iRMX 86 Development Procedures
- Chapter 8. Files Utility System

The minimum hardware requirements for custom boards that utilize the iAPX 86 microprocessor for an iRMX 86 application system are defined in Appendix C.

The manual offers the following information:

- Introduces you to the Operating System and shows how it is packaged.
- Defines the hardware and software environment in which application systems are developed.
- Describes how to perform the required hardware modifications to support the iRMX 86 Operating System, and shows how to assemble the various hardware pieces.
- Explains how to install and use the demonstration system. This is a tested and debugged iRMX 86 application system.
- Describes the Start-Up System and Files Utility and how to use these subsystems to format iRMX 86 disks, create and delete files, and transfer information between ISIS-II disks and iRMX 86 disks.
- Describes how to use the Patching Utility to replace obsolete or flawed relocatable object modules by overlaying Intel-supplied or user-created replacement modules over the old code.
- Outlines the process for developing your own iRMX 86-based application system.

READER LEVEL

This manual assumes that you are already familiar with the following Intel software and hardware elements:

- The INTELLEC Microcomputer Development System and the ISIS-II Operating System
- The ASM86 Macro Assembly Language and/or the PL/M-86 programming language
- Either the ICE-86 In-Circuit Emulator or the iSBC 957A/B Interface and Execution Package
- The individual hardware components that make up an iRMX 86 target system

It is also assumed that you have read the INTRODUCTION TO THE iRMX 86 OPERATING SYSTEM manual.

RELATED PUBLICATIONS

The following manuals provide additional background and reference information:

<u>Manual</u>	<u>Number</u>
Introduction to the iRMX™ 86 Operating System	9803124
iRMX™ 86 Configuration Guide	9803126
iRMX™ 86 Nucleus Reference Manual	9803122
iRMX™ 86 Terminal Handler Reference Manual	143324
iRMX™ 86 Debugger Reference Manual	143323
iRMX™ 86 Basic I/O System Reference Manual	9803123
iRMX™ 86 Extended I/O System Reference Manual	143308
Guide to Writing Device Drivers for the iRMX™ 86 and iRMX 88™ I/O Systems	142926
iRMX™ 86 Loader Reference Manual	143318
iRMX™ 86 Human Interface Reference Manual	9803202
iRMX™ 86 System Programmer's Reference Manual	142721
iRMX™ 86 Programming Techniques	142982
ISIS-II User's Guide	9800306
INTELLEC Series III Microcomputer Development System Console Operating Instructions	121609
ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users	9800714
iSBC™ 957A INTELLEC -- iSBC™ 86/12A Interface and Execution Package User's Guide	142849
Users Guide for the iSBC™ 957B iAPX 86, 88 Interface and Execution Package	143979
iSBC™ 86/12A Single Board Computer Hardware Reference Manual	9803074
iSBC™ 86/05 Single Board Computer Hardware Reference Manual	143153
iSBC™ 88/25™ Single Board Computer Hardware Reference Manual	143825

RELATED PUBLICATIONS (continued)

<u>Manual</u>	<u>Number</u>
iSBC™ 88/40 Measurement and Control Computer Hardware Reference Manual	142978
iSBC™ 337 Multimodule™ Numeric Data Processor Hardware Reference Manual	142887
iCS™ 80 Industrial Chassis Hardware Reference Manual	9800799
iSBC™ 660 System Chassis Hardware Reference Manual	9800505
iSBC™ 680/681™ Multistore™ User System Package Hardware Reference Manual	162432
iSBC™ 204 Flexible Diskette Controller Hardware Reference Manual	9800568
iSBC™ 206 Disk Controller Hardware Reference Manual	9800567
iSBC™ 208 Flexible Disk Controller Hardware Reference Manual	143078
iSBC™ 215 Winchester Disk Controller Hardware Reference Manual	121593
iSBX™ 218 Flexible Disk Controller Hardware Reference Manual	121583
iSBC™ 220 SMD Disk Controller Hardware Reference Manual	121597
iSBC™ 254 Bubble Memory Technical Manual	112179
iSBC™ 032/048/064 Random Access Memory Boards Hardware Reference Manual	9800488
iSBC™ 016A/032A/064A/028A/056A™ RAM Boards Hardware Reference Manual	143572
8086/8087/8088 Macro Assembly Language Reference Manual for 8086-Based Development Systems	121627
8086/8087/8088 Macro Assembler Operating Instructions	121628
ASM86 Language Reference Manual	121703

CONTENTS

	PAGE
CHAPTER 1	
INTRODUCTION TO THE iRMX 86 PACKAGE	
Inventory.....	1-2
Recommendations.....	1-3
CHAPTER 2	
iRMX 86 DEVELOPMENT ENVIRONMENT	
General Requirements.....	2-2
Development System.....	2-2
Target System.....	2-2
Application-Dependent Requirements.....	2-3
CHAPTER 3	
HARDWARE CONSIDERATIONS	
Board Modifications.....	3-1
iSBC 86/12A Board Modifications.....	3-2
iSBC 204 Board Modifications.....	3-3
iSBC 206 Board Modifications.....	3-4
iSBC 208 Board Modifications.....	3-5
iSBC 215 and iSBX 218 Board Modifications.....	3-5
iSBC 220 Board Modifications.....	3-6
iSBC 254 Board Modifications.....	3-6
Memory Board Jumper Connections.....	3-6
Chassis Board Arrangement.....	3-6
Cable Connections.....	3-7
Logical and Physical Device Names.....	3-8
CHAPTER 4	
NUCLEUS DEMONSTRATION SYSTEM	
Hardware Requirements.....	4-1
Loading the Nucleus Demonstration System.....	4-2
Using the Nucleus Demonstration System.....	4-3
Operating Modes.....	4-3
Changing Statement Lines.....	4-4
Variables.....	4-4
Constants.....	4-5
Expressions.....	4-5
Statements and Functions.....	4-5
Basic Statements.....	4-7
FOR ... NEXT.....	4-7
GOSUB.....	4-8
GOTO.....	4-8
IF.....	4-8
INPUT.....	4-9
LET.....	4-9
LIST.....	4-10
NEW.....	4-10

CONTENTS (continued)

	PAGE
CHAPTER 4	
DEMONSTRATION SYSTEM	
Basic Statements (continued)	
PRINT.....	4-10
REM.....	4-11
RETURN.....	4-11
RUN.....	4-11
STOP.....	4-11
Basic Functions.....	4-11
ABS.....	4-11
RND.....	4-12
SIZE.....	4-12
iRMK 86 Statements and Functions.....	4-12
CATALOG.....	4-13
CRTMBOX.....	4-13
CRTSEGM.....	4-14
CRTSEMA.....	4-14
CRTTASK.....	4-15
DELMBOX.....	4-16
DELSEGM.....	4-16
DELSEMA.....	4-17
DELTASK.....	4-17
GETTKNS.....	4-17
LOOPKUPO.....	4-18
RCVUNIT.....	4-19
RECVMSG.....	4-19
RESTASK.....	4-20
SENDMSG.....	4-21
SLEEP.....	4-21
SNDUNIT.....	4-22
SUSTASK.....	4-22
UNCATLG.....	4-23
 CHAPTER 5	
START-UP SYSTEM	
Functions Provided.....	5-1
Hardware Required.....	5-2
Using the Start-Up System.....	5-2
Getting Started.....	5-2
Start-Up System Commands.....	5-4
File Management Commands.....	5-4
Additional Services.....	5-5
Special Use Restrictions.....	5-5

CONTENTS (continued)

	PAGE
CHAPTER 6	
PATCHING UTILITY	
Invoking the Patching Utility.....	6-2
Patching Procedures.....	6-3
Jump Instruction Patch.....	6-3
In-Place Patch.....	6-4
Patching Library Modules.....	6-5
Listing Translator Header Records.....	6-6
Error Messages.....	6-5
CHAPTER 7	
iRMX 86 DEVELOPMENT PROCEDURES.....	7-1
CHAPTER 8	
FILES UTILITY SYSTEM	
Functions Provided.....	8-1
Hardware Required.....	8-1
Starting the Files Utility.....	8-2
Using the Files Utility.....	8-3
Changing Diskettes.....	8-3
Commands.....	8-3
ATTACHDEV.....	8-3
BREAK.....	8-4
CREATEDIR.....	8-4
DELETE.....	8-4
DETACH.....	8-5
DIR.....	8-5
DOWNCOPY.....	8-5
FORMAT.....	8-6
HELP.....	8-7
UPCOPY.....	8-8
Error Messages.....	8-8
APPENDIX A	
ORIGINAL BOARD JUMPER CONNECTIONS.....	A-1
APPENDIX B	
iRMX 86 CONDITION CODES SUMMARY.....	B-1
APPENDIX C	
HARDWARE REQUIREMENTS FOR CUSTOM CONFIGURATIONS.....	C-1
APPENDIX D	
iRMX 86 SOFTWARE VERSION NUMBERS.....	D-1

FIGURES

	PAGE
2-1. iRMX 86™ Development Environment Example.....	2-1

TABLES

2-1. Memory Requirements.....	2-4
3-1. Parallel I/O Connector J1 Pin Assignments.....	3-3
3-2. iRMX 86 Physical Device Names.....	3-9
4-1. Statement and Function Dictionary.....	4-6
A-1. Original iSBC 86/12A Jumpers.....	A-1
A-2. Original iSBC 204 Jumpers.....	A-2
A-3. Original iSBC 206 Pin Connections (Channel Board).....	A-2
A-4. Original iSBC 208 Jumpers.....	A-2
A-5. Original iSBC 215A Jumpers.....	A-2
A-6. Original iSBC 215B Jumpers.....	A-3
A-7. Original iSBC 218 Jumpers.....	A-3
A-8. Original iSBC 220 Jumpers.....	A-3
A-9. Original iSBC 254 Pin Connections.....	A-3
B-1. iRMX 86 Condition Codes.....	B-1
D-1. iRMX 86 Software Version Numbers.....	D-1

CHAPTER 1. INTRODUCTION TO THE iRMX™ 86 PACKAGE

The iRMX 86 Operating System is a real-time, multitasking operating system for iAPX 86,88-based microcomputers. The system consists of a Nucleus and various optional subsystems, as follows:

- Nucleus -- the central control element of the Operating System. It coordinates system activities.
- Terminal Handler -- provides the interface between an executing application program and the terminal.
- Basic I/O System -- provides generalized but powerful file and device access capabilities, while making few or no assumptions about an application's specific requirements.
- Extended I/O System -- features ease-of-use, buffering, synchronization, and shorter parameter lists for system calls.
- Human Interface -- provides an interactive command set for performing file management and various utility functions. Also provides a set of system calls that expedite creation of new non-resident application programs that can be loaded and executed by keyboard commands.
- Application Loader -- loads absolute files, load-time locatable files, and position-independent code files into memory from secondary storage.
- Bootstrap Loader -- loads executable modules into memory at system startup.
- Debugger -- provides monitoring and debugging capabilities during program development.
- Other available software includes the Start-Up System, a Files Utility system, a Patching Utility, and a Universal Development System Interface.

The software that you write runs under the supervision of the Nucleus and in conjunction with any desired optional subsystems.

When you receive them from Intel, the iRMX 86 Nucleus and other Operating System software modules reside on diskettes as relocatable libraries. You use an INTELLEC Development System to combine this iRMX 86 code with your application code to produce object code that executes on an iAPX 86-based microcomputer. The end result is an iRMX 86 application system that is configured to your specific requirements.

INTRODUCTION TO THE iRMX™ 86 PACKAGE

INVENTORY

Your shipment of iRMX 86 materials includes fourteen manuals (including this one) and ten diskettes. The manuals are:

- iRMX 86 INSTALLATION GUIDE -- This manual which you are now reading helps you to make specific iRMX 86-required hardware modifications, install and run the demonstration system, and use the iRMX 86 Start-Up System, Files Utility, and Patching Utility.
- INTRODUCTION TO THE iRMX 86 OPERATING SYSTEM -- This manual introduces you to the iRMX 86 product. Read this manual before any other in the supplied set.
- iRMX 86 NUCLEUS REFERENCE MANUAL -- This manual is the primary reference source for the Nucleus. Knowledge of the Nucleus architecture and its interaction with other system modules is essential.
- iRMX 86 TERMINAL HANDLER REFERENCE MANUAL -- This manual contains both operator instructions and programming information for iRMX 86 systems that use a terminal.
- iRMX 86 DEBUGGER REFERENCE MANUAL -- This manual describes the use and capabilities of the iRMX 86 Debugger.
- iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL -- This manual is the primary reference source for the Basic I/O System. Both this manual and the iRMX EXTENDED I/O SYSTEM REFERENCE MANUAL should be studied before you come to a decision as to which I/O system best meets your application requirements.
- iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL -- This manual is the primary reference source for the Extended I/O System. Since the Extended I/O System is a superset of the Basic I/O System, you should also read the the BASIC I/O SYSTEM REFERENCE MANUAL (the Basic I/O System calls may be ignored) before installing the Extended I/O System on your configuration.
- iRMX 86 LOADER REFERENCE MANUAL -- This manual describes the Bootstrap Loader, which can load and start iRMX 86 application systems, and the Application Loader, which enables application systems to perform loading under iRMX 86 control.
- iRMX 86 HUMAN INTERFACE REFERENCE MANUAL -- This manual is the primary reference source both for using the Human Interface commands to perform file management and utility functions, and for using the Human Interface system calls to write new applications that can be invoked by interactive keyboard commands.
- iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL - This manual describes selected features of the Operating System, not covered in the other manuals, which are for use by system programmers only.

INTRODUCTION TO THE iRMX™ 86 PACKAGE

- iRMX 86 PROGRAMMING TECHNIQUES - This manual provides system and application programmers with techniques that reduce development time.
- GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O SYSTEMS - This manual describes how to write device drivers that interface with the iRMX 86 I/O System.
- iRMX 86 CONFIGURATION GUIDE - This manual describes how to build a software application system by combining the Operating System and application software.
- iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL - This manual describes how to use the disk verification utility and describes the structure of an iRMX 86 volume that contains named files.

Nine of the ten diskettes in your iRMX 86 package will be in either double-density or single-density ISIS-II format, depending on which version you specified when you ordered your system (each version has its own Product Order Number). Both sets have the same contents. The remaining diskette, that containing the Start-Up System, is single-density, with iRMX 86 format.

The nine ISIS-II diskettes included in either set are as follows:

- Nucleus diskette
- Terminal Handler, Debugger diskette
- Basic I/O System diskette
- Extended I/O System diskette
- Loader diskette (Application and Bootstrap Loaders)
- Human Interface diskette
- Nucleus Demonstration diskette
- Utilities diskette
- Universal Development System Interface diskette

RECOMMENDATIONS

To prevent the possibility of accidentally destroying system software, you should make at least one backup copy of each ISIS-II and iRMX 86 diskette that you are planning to use. Keep the Intel-supplied diskettes as masters. Use the copies for system development.

CHAPTER 2. iRMX™ 86 DEVELOPMENT ENVIRONMENT

The development of an iRMX 86-based application system requires several hardware and software components. Some of these components are always required and others are a function of the particular application system. Figure 2-1 shows a typical development hardware environment.

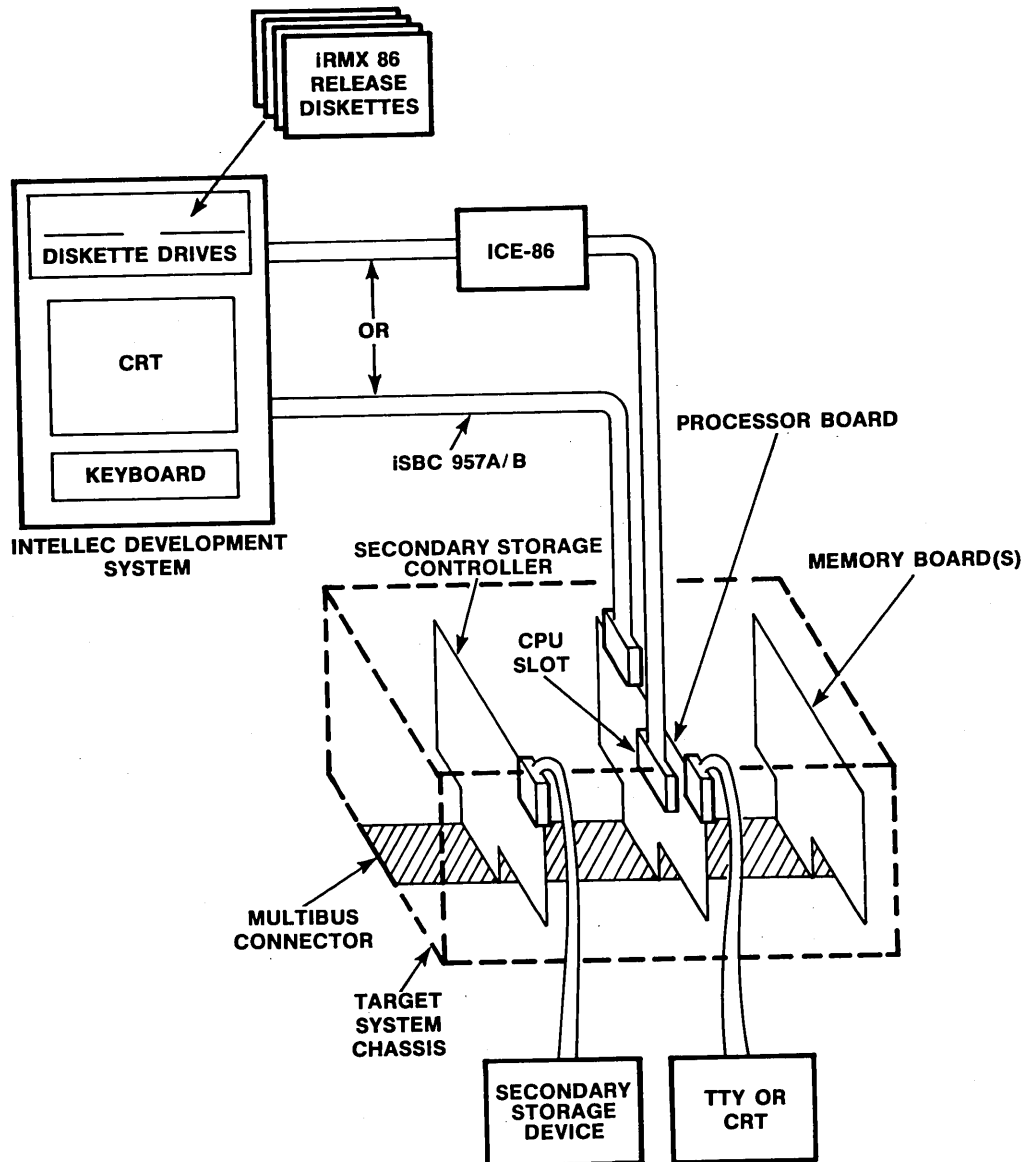


Figure 2-1. iRMX™ 86 Development Environment Example

Figure 2-1 illustrates the interface between the INTELLEC Development System, where the software is developed, and the target system where the application system actually runs. The figure shows devices that are commonly attached to the target system. However, you can also attach other available devices.

GENERAL REQUIREMENTS

The following items are required in the development of any iRMX 86-based application system.

DEVELOPMENT SYSTEM

- An INTELLEC Development System with CRT, keyboard, at least four disk drives, and 64K bytes of RAM.
- A diskette containing the ISIS-II Operating System.
- An ASM86 Macro Assembler and/or PL/M-86 compiler, as well as the ASM86 Software Development Utilities, on diskette.
- Diskettes containing the Intel-supplied iRMX 86 software.

TARGET SYSTEM

- An iAPX 86,88-based microprocessor, which is the basic element of the application system. You can use 8086, 8284A, 8259A, 8253, and 8251A components for your target system. However, the iSBC 86/12A board is suggested for prototyping and debugging.
- A chassis to supply power to the processor board and any other system boards.
- Enough memory to contain the Nucleus, selected subsystems, and your application jobs.
- If your application uses the Application Loader, the Human Interface, or the Bootstrap Loader, you must have secondary storage device(s) and appropriate controllers.

You configure all of your software with the Development System and then transfer it to the target system for execution. You can use any of the following products to transfer code to the target system RAM:

- The ICE-86 In-Circuit Emulator
- The iSBC 957A/B Interface and Execution Package
- The Bootstrap Loader

iRMX™ 86 DEVELOPMENT ENVIRONMENT

The ICE-86 In-Circuit Emulator and the iSBC 957A/B Interface and Execution Package transfer code from an ISIS-II diskette on the INTELLEC Development System, while the Bootstrap loader transfers code from an iRMX 86 diskette in a target system drive.

After you have tested the code, you can burn it into PROM and place the PROM on the target system to eliminate using the ICE-86 emulator or the iSBC 957A/B package to load the code.

If you do not wish to burn the code into PROM or ROM, use either the Start-Up System or the Files Utility with the iSBC 957A/B package to place your tested code on an iRMX 86 formatted diskette or in bubble memory, and then use the Bootstrap Loader to load the code directly into memory.

APPLICATION-DEPENDENT REQUIREMENTS

You may need additional hardware for your target system, depending on your application requirements:

- If your application includes the Terminal Handler or the Debugger, connect an RS232 interface terminal to the serial I/O port of the iSBC 86/12A board.
- If your application includes the I/O System and you intend to use named or physical files, place at least one controller board in the chassis with the iSBC 86/12A board. You can use any of the following controller boards:

- iSBC 204
- iSBC 206
- iSBC 208
- iSBC 215A or iSBC 215B
- iSBX 218
- iSBC 220
- iSBC 254

Connect the controllers to their associated secondary storage devices. If only stream files are used, the I/O System can be used without a controller board.

If your application system includes the I/O System and you are using disks, you must also use the Start-Up System, the Files Utility, or the Human Interface subsystem to format disks for use in your target system disk drives. These systems are described later in this manual.

Target system memory requirements vary, depending on the type of software included. Table 2-1 lists the maximum memory requirements for Intel-supplied products. These requirements are divided into ROM and RAM requirements; however, if you first test your system in RAM, RAM must be large enough to satisfy all memory requirements.

Table 2-1. Memory Requirements

	Code and Constants				Static-Data Memory	Dynamic-Data Memory
	Minimum		Maximum			
	Size	Support	Size	Support		
Nucleus	12.6K	N/A	24.3K	N/A	2 Bytes	1.9K*
Terminal Handler	2.7K	14.5K	2.7K	14.8K	.2K	1.4K
Debugger	28.3K	15.8K	28.3K	16.1K	1.2K	6.0K
Basic I/O System	24.5K	14.6K	49.0K	19.6K	.1K	2.8K
Extended I/O System	7.3K	64.0K	11.6K	69.7K	.1K	1.5K
Application Loader	3.9K	64.0K	9.0K	79.4K	.1K	.5K
Bootstrap Loader	.4K	N/A	1.9K	N/A	6.4K***	N/A
Human Interface	15.8K**	90.8K	16.5K**	91.1K	.1K	8.0K
<p>* Does not include memory required for the user-defined Root Object Directory.</p> <p>** Does not include non-resident commands which range in size from 3K to 12K.</p> <p>*** Available to Nucleus for allocation as Dynamic-Data Memory.</p>						

Explanation of Headings and Symbols:

Code and Constants: Memory required to hold code and the constants associated with code. This memory may be ROM, PROM, or RAM

Static-Data Memory: Memory required to hold variables associated with code. This memory is not dynamically allocated by the operating system and must be RAM.

Dynamic-Data Memory: Memory required to execute the code. This memory is dynamically allocated during operating system initialization and must be RAM.

Support: The minimum number of bytes of other iRMX 86 components required to support the indicated iRMX 86 component.

K: 1024 bytes.

iRMX™ 86 DEVELOPMENT ENVIRONMENT

There will also be application-related memory in your system. This memory is required for application code, constants, variables, and dynamically allocated structures. The latter includes both memory explicitly allocated by the application and memory implicitly allocated as the result of calls to iRMX 86 functions. This memory is divided between RAM and ROM.

CHAPTER 3. HARDWARE CONSIDERATIONS

This chapter describes the hardware modifications and installation procedures that apply specifically to iRMX 86 systems that are to run on iSBC 86/12A boards. Topics covered include the following processes:

- Modifying the iSBC 86/12A board and other associated boards by connecting jumper posts and setting switches.
- Arranging the boards in the chassis.
- Installing the cables.

Although this chapter contains instructions on how to modify hardware components, it is primarily useful in regard to the modifications needed to enable the Start-Up System or the Files Utility to run. It will also allow you to use the line printer driver. Furthermore, it is assumed that the boards have not previously been modified; that is, that the factory jumpers, and only the factory jumpers, are in place. Appendix A contains a list of factory-installed jumpers for each of the iSBC 86/12A, 204, 206, 208, 215A, 215B, 220, 254 and iSBX 218 boards.

You might have to make additional changes to support specific hardware, such as a particular brand of disk drive. When you begin development, your jumpering requirements will depend upon your application. Information of this type is not included here but can be found in the appropriate hardware reference manuals.

BOARD MODIFICATIONS

Before you insert your iSBC 86/12A board, controller boards, and memory boards into the chassis, you must modify certain jumper connections and switch settings on these boards. The following sections describe the modification process. Each section on a controller board assumes that you are using only that type of controller with the iSBC 86/12A board.

HARDWARE CONSIDERATIONS

iSBC 86/12A BOARD MODIFICATIONS

Jumpering is the means whereby interrupt levels are assigned to their various purposes. The Start-Up System's Terminal Handler uses master level 6 for reading and master level 7 for writing; controllers use master levels 3, 4, and 5; and the system clock uses level 2. To support these interrupt assignments, make all of the following jumper connections:

<u>Level</u>	<u>Connection</u>
2	79-83 (default)
3	70-78
4	69-77
5	68-76 (default)
6	75-82
7	74-90

Most of the jumpering necessary to support timing and the USART is done at the factory. The only additional jumper that is needed is 51-52.

To place the dual port RAM in the correct 128K byte area of the 1-megabyte address space, remove jumper 125-126 and install jumper 127-128.

The switch settings on switch S1 must be set as follows:

<u>Switch</u>	<u>Setting</u>
1	ON
2	OFF with iSBX 300 multimodule; otherwise, ON
3	OFF
4	OFF
5	OFF
6	OFF
7	ON
8	OFF

Additional jumper changes are required if you are using either the iSBC 957A or the iSBC 957B package. The iSBC 957A package requires that the factory-installed jumpers 94-96 and 97-98 be in place. The iSBC 957B package requires that the factory installed jumpers 94-96 be in place, that the jumper 97-98 be removed, and that jumper 97-99 be installed. If sockets A28, A29, A46, and A47 of your iSBC 86/12A board contain memory other than the iSBC 957A or iSBC 957B PROMs, consult the iSBC 86/12A SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL for information regarding the setting of switches 7 and 8 and the jumpering of posts 94 through 99.

Jumper posts 7 through 34 are related to parallel port I/O configuration and, in general, can be connected as you wish. However, if you are using the iSBC 957A/B package, you must use it with the parallel port and make the appropriate jumpering connections as described in either the iSBC 957A INTELLEC -- iSBC 86/12A INTERFACE AND EXECUTION PACKAGE USER'S GUIDE or the USER'S GUIDE FOR THE iSBC 957B iAPX 86,88 INTERFACE AND EXECUTION PACKAGE.

Jumper 12-21 must not be installed if there is an iSBC 337 multimodule on the iSBC 86/12A board. Otherwise, the jumper 12-21 (ground to TEST/pin) must be in place. Master interrupt level 0 is recommended for an

HARDWARE CONSIDERATIONS

iSBC 337. Use interrupt pin E81 to support this level. Refer to the iSBC 337 MULTIMODULE NUMERIC DATA PROCESSOR HARDWARE REFERENCE MANUAL for a description of the installation procedure.

All of the remaining factory-installed jumpers are appropriate for use with the Start-Up System and the Files Utility. You may change these jumpers to meet special conditions, such as an unusual bus priority resolution scheme.

Making the changes that allow the use of the line printer driver may be one such special condition. To use the line printer driver, the following changes must be made to the iSBC 86/12A board.

Master interrupt level 1 is used by the line printer. To support this interrupt level connect jumpers E84 and E80. It is also necessary to remove the jumpers E13-E14 and E32-E33 and install a jumper E22-E32. You will also need to plug the driver IC, 7438 (P.N. 100908-001), into location A11 and the terminator packs, iSBC 902 (P.N. 4500645-01), into locations A10, A12, and A13. The parallel connection to the iSBC 957A/B monitor can not be used if the line printer is used - so whatever changes were made for parallel port usage will have to be returned to the original factory settings.

The line printer driver interfaces the Basic I/O system physical file driver to the 8255 parallel I/O port connected to a Centronix type line printer. In order to connect the iSBC 86/12A board and the Centronix type printer, a cable with the following pin assignments must be used.

Table 3-1. Parallel I/O Connector J1 Pin Assignments

Pin	Function	Pin	Function
23	Character strobe to printer	37	Data Bit 5
25	Printer ready (online)	39	Data Bit 4
27	Paper error status	41	Data Bit 3
29	Character acknowledge from printer	43	Data Bit 2
33	Data Bit 7	45	Data Bit 1
35	Data Bit 6	47	Data Bit 0

All even pin numbers are tied to ground as logic ground returns.

iSBC 204 BOARD MODIFICATIONS

The iRMX 86 software expects interrupts from the iSBC 204 board to come in at level 5. To prepare for this, remove jumper 63-67 (for level 2) and install jumper 67-71.

HARDWARE CONSIDERATIONS

Select an iSBC 204 base address of 0A0H by setting the S2 switches as follows:

<u>Switch</u>	<u>Setting</u>
4	OFF
5	ON
6	OFF
7	ON

Because the Basic I/O System configuration for the Start-Up System and Files Utility supports only 8-inch diskette devices, you must leave jumpers 23-24 and 37-38 in place.

No other jumpering is required, but you might want to expand the port address availability of the iSBC 204 board on the MULTIBUS interface by setting it for 12-bit addressing. This is done by removing jumper B-C from pad W1 and replacing it with jumper A-B.

If your iSBC 204 board has two 8271 Floppy Disk Controller chips, remove jumpers 75-76 and 77-78.

iSBC 206 BOARD MODIFICATIONS

Set the iSBC 206 channel board to interrupt the iSBC 86/12A board at level 4. This is done by setting rotary switch S2 to position 4.

The required base address of 68H is set by making the following S1 switch settings on the channel board:

<u>Switch</u>	<u>Setting</u>
1	OFF
2	OFF
3	ON
4	OFF
5	OFF
6	ON
7	ON
8	OFF

The settings of switches SW1 and SW2 on the interface board are device-dependent. The physical names of your devices depend upon the size of the sectors on the devices. If you use 512-byte sectors, the physical name of the device is D0, whereas if you use 128-byte sectors, the physical name of the device is DS0. (Table 3-2 contains complete information concerning physical device names.) The SW2 switch settings are as follows:

<u>Switch</u>	<u>Sector Size</u>	<u>Setting</u>
1 to 4	128 or 512	OFF
5 to 8	512	ON
5 to 8	128	OFF

HARDWARE CONSIDERATIONS

iSBC 208 BOARD MODIFICATIONS

The iRMX 86 software expects interrupts from the iSBC 208 board to come in at level 5. To prepare for this, install a jumper from E79 to E84.

Select an iSBC 208 base address of 0000H. This is the way it comes from the factory.

You should expand the port address availability of the iSBC 208 board on the MULTIBUS interface by setting it for 16-bit addressing. This is done by removing the shorting plug connecting E45 and E49 and installing the plug between E41 and E45.

Because the Basic I/O System configuration for the Start-Up System and Files Utility support only 8-inch diskette devices, terminal posts E4 and E5 must be left in the standard factory configuration (jumper link removed).

iSBC 215 AND iSBX 218 BOARD MODIFICATIONS

Your iSBC 215 board should be set for level 4 interrupts. This is done by removing jumper C-5 (for level 5) on pad W19 and installing jumper C-4.

The wake-up address of 0100H is set at switches S1 and S2. The S1 switches 1 through 7 should be OFF, as should the S2 switches 3 through 10. S1 switch 8 should be ON.

To provide 16-bit bus compatibility, S2 switches 1 and 2 should be ON and the following jumper connections should be made:

<u>Pad</u>	<u>Connection</u>
W18	1-2
W20	1-3
W21	1-3

The jumpering of pads W1, W2, W5 through W10, W13 through W17, and W22 depends upon the type of Winchester disk device being used with the iSBC 215 board. Refer to the iSBC 215 WINCHESTER DISK CONTROLLER HARDWARE REFERENCE MANUAL for instructions as to these jumpers.

The iSBX 218 multimodule must be plugged into socket J4 of the iSBC 215 board. Related to this, the default jumpering on pads W3, W4, W11, and W12 is appropriate for the iRMX 86 environment, except that you must install jumper 1-3 on pad W11.

On the iSBX 218 board, you must install jumper A-B on pad W1 to support direct memory access. In addition, install jumper A-C on each of the pads W3 through W7 to support 8-inch drives.

HARDWARE CONSIDERATIONS

iSBC 220 BOARD MODIFICATIONS

Your iSBC 220 board should be set for level 4 interrupts. This is done by installing the jumper link between W4-c and W4-4.

The wake-up address of 0100H is set at switches S1 and S2. The S1 switches 1 through 7 should be OFF, as should the S2 switches 3 through 10. S1 switch 8 should be ON. To provide 16-bit bus compatibility, S2 switches 1 and 2 should be ON.

iSBC 254 BOARD MODIFICATIONS

Set your iSBC 254 board for level 3 interrupts. Do this by removing jumpers 67-68 and 75-76, and installing jumper 73-74. Later, when you prepare your hardware for purposes other than running the Start-Up System or Files Utility, use interrupt level 0, 1, 2, or 3 with this board.

The board must be set up for a base address of 800H, with 12-bit addressing. To arrange this, remove jumper 45-46, install jumper 47-48, and leave jumper 29-30 installed.

MEMORY BOARD JUMPER CONNECTIONS

Connect the jumpers on your iSBC 032/048/064 RAM boards or your 016A/032A/064A/028A/056A Ram boards to assign memory to the correct memory locations. Refer to the iSBC 032/048/064 RANDOM ACCESS MEMORY BOARDS HARDWARE REFERENCE MANUAL or the iSBC 016A/032A/064A/028A/056A RAM BOARDS HARDWARE REFERENCE MANUAL for the procedure.

CHASSIS BOARD ARRANGEMENT

A typical hardware environment for iRMX 86 applications includes several master boards, which contend for control of the MULTIBUS interface. In order to prevent conflict, it is necessary (even if you have only one master board) to assign a bus contention priority to each master board in your system. The possible master boards in an iRMX 86 environment are the iSBC 86/12A, 204, 206 channel, 208, 215A, 215B, 220, and 254 boards. Two priority schemes are available: serial, which supports up to three bus masters; and parallel, which supports an unlimited number of bus masters but is more difficult to implement.

The following paragraphs assume the use of an iSBC 604 Cardcage/-Backplane, optionally with an iSBC 614 Expansion Cardcage/Backplane. These cardcage/backplanes are used in the iSBC 655, iSBC 660, and ICS 80 chassis.

HARDWARE CONSIDERATIONS

In the serial priority scheme, the top slot (J2) of the cardcage has the highest priority, and the rest of the slots have progressively lower priorities, with slot J5's priority being the lowest. To implement the serial priority scheme, ground the BPRN/ signal of the highest priority bus master by jumpering between posts B and N on an iSBC 604 Cardcage/Backplane or between posts B and L on an iSBC 614 Expansion Cardcage/Backplane. If you need to leave a slot empty, jumper the priority signal around the slot, as described in the iSBC 86/12A SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL.

If you find it necessary to use the parallel priority scheme, the procedure for implementing it can be found in the iSBC 86/12A SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL, and also in the reference manuals for the iSBC 660 and iCS 80 chassis.

The order in which you arrange the boards in the cardcage depends upon the type and number of bus masters in your system and upon their physical placement requirements. Physical requirements include placing the iSBC 206 channel and interface boards in adjacent slots and, if you are using the ICE-86 In-Circuit Emulator, placing the iSBC 86/12A board in the top slot. Moreover, if you are using an ICE-86 emulator with an iCS 80 chassis, you will need an extender card in the top slot.

In general, the following guidelines can help you assign priorities to bus masters:

- The iSBC 204 board lacks the ability to wait for access to the MULTIBUS interface, so it should be highest in priority.
- The iSBC 208 board lacks the ability to wait long for access to the MULTIBUS, so it should occupy a high priority slot.
- The iSBC 254 board has only limited ability to wait for access, so it should be high in priority.
- The iSBC 206, 215 (including the iSBX 218), and 220 boards can wait for access, so they need not be high in priority.
- The iSBC 86/12A board should be lowest in priority.

You should be prepared to experiment with various board arrangements in the chassis, perhaps even to violate these guidelines.

CABLE CONNECTIONS

After you have placed the boards in the chassis, install the cables that join the boards to various parts of the system. If you are using the Terminal Handler, Debugger, I/O System Terminal Driver, or Start-Up System, connect the terminal of your iRMX 86 system to the serial I/O port of the iSBC 86/12A board. The iSBC 86/12A SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL describes the procedure.

HARDWARE CONSIDERATIONS

If you are using the I/O System with secondary storage devices, connect the drives to their associated I/O connectors on the controllers boards.

Connect the cables for the ICE-86 In-Circuit Emulator or the iSBC 957A/B package, depending on which package you use to load your software. If you use the ICE-86 Emulator to load your software, install the ICE-86 boards in the Development System chassis and connect the ICE-86 cable to the 8086 socket on the iSBC 86/12A board. Refer to the ICE-86 IN-CIRCUIT EMULATOR OPERATING INSTRUCTIONS FOR ISIS-II USERS for a description of this process.

If you use the iSBC 957A/B package to load your software, install the PROM set on the iSBC 86/12A board and use the cable to connect the UPP output on the Development System to the parallel I/O port on the iSBC 86/12A board. If you are not using the target system terminal, you can connect to either the serial or the parallel I/O port on the iSBC 86/12A board. Refer to the iSBC 957A INTELLEC -- iSBC 86/12A INTERFACE AND EXECUTION PACKAGE USER'S GUIDE or the USER'S GUIDE FOR THE iSBC 957B iAPX 86,88 INTERFACE AND EXECUTION PACKAGE for a description of this process.

LOGICAL AND PHYSICAL DEVICE NAMES

When using the Start-Up System or the Files Utility, you reference files by means of a device name/file name (or path name) combination. The device name is a "logical" name of your choice that is assigned when the device is being attached (ATTACHDEV in the Files Utility and ATTACHDEVICE in the Start-Up System). Before these assignments are made, the iRMX 86 Operating System knows devices by their "physical" names, which are given in Table 3-2 for each applicable device. These physical names are associated with logical names by means of an attach device command.

Suppose, for example, that you want to copy the file JOBA from an ISIS-II diskette on unit 1 of an INTELLEC Development System to a file of the same name on an iRMX 86 diskette on unit 0 of a Shugart SA800 drive interfaced with an iSBC 204 controller that is jumpered for 128-byte sectors. Suppose further that the Shugart device (whose physical name is F0) has been given the logical name :DRIVE3:. To initiate the copy operation with the Files Utility, enter the following:

```
UPCOPY :F1:JOBA TO :DRIVE3:JOBA
```

HARDWARE CONSIDERATIONS

Table 3-2. iRMX™ 86 Physical Device Names

Device Names	Device Type	Unit Number	Sides	Density	Bytes per Sector
Flexible Disk Drives					
FO	204 Shugart SA800	0	1	Single	128
F1	204 Shugart SA800	1	1	Single	128
FX0	204 Shugart SA800	0	1	Single	512
FX1	204 Shugart SA800	1	1	Single	512
AFO	208 Shugart SA800	0	1	Single	128
AF1	208 Shugart SA800	1	1	Single	128
AFDO	208 Shugart SA800	0	1	Double	256
AFD1	208 Shugart SA800	1	1	Double	256
AMFO	208 Shugart SA410	0	1	Double	256
AMF1	208 Shugart SA410	1	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
WFO	218 Shugart SA800	0	1	Single	128
WF1	218 Shugart SA800	1	1	Single	128
WFDO	218 Shugart SA800	0	1	Double	256
WFD1	218 Shugart SA800	1	1	Double	256
WMFO	218 Shugart SA410	0	1	Double	256
WMF1	218 Shugart SA410	1	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
Hard Disk Drives					
D0	206	0			512
D1	206	1			512
DS0	206	0			128
DS1	206	1			128

HARDWARE CONSIDERATIONS

Table 3-2. iRMX™ 86 Physical Device Names (continued)

Device Names	Device Type	Unit Number	Sides	Density	Bytes per Sector
Winchester Disk Drives					
IWO	215 Priam 3450				1024
MWO	215 Memorex 101				1024
PWO	215 Pertec D8000				1024
SWO	215 Shugart SA1002				1024
Storage Module Disk Drives					
SMD0	220	0			1024
SMD1	220	1			1024
Bubble Memory Drive					
B0	254, 4 bubbles				256

In addition, the byte bucket has the physical device name BB, and the terminal has the name T0.

CHAPTER 4. NUCLEUS DEMONSTRATION SYSTEM

The Nucleus demonstration system is contained on one of the diskettes shipped with the iRMX 86 package. It can be used to familiarize yourself with system operation and to test system performance.

The Nucleus demonstration diskette contains a complete configuration module for an application system. This application system consists of the Nucleus, the Debugger, the root job, and an application job called TBASIC, a BASIC interpreter which allows you to write programs that manipulate iRMX 86 objects. This chapter describes what hardware and software you need to run the Nucleus demonstration system, how to load the system, and how to use it.

HARDWARE REQUIREMENTS

You need the following equipment to run the Nucleus demonstration system:

- o An INTELLEC Microcomputer Development System with CRT, keyboard, and at least two disk drives
- o An iSBC 86/12A board and chassis
- o 128K of contiguous RAM, starting at address 0, for use in the target system
- o A RS232 protocol terminal
- o Either the ICE-86 In-Circuit Emulator or the iSBC 957A/B package

You need the terminal in order to communicate with the application system, and either the ICE-86 emulator or the iSBC 957A/B package to load the system from diskette to memory.

Since you are using the Debugger in this system, make sure that the iSBC 86/12A board is jumpered to support the interrupt levels that the terminal uses.

The Nucleus demonstration system assumes that your terminal operates at 9600 baud. If it operates at a different baud rate, you must reconfigure the Debugger and specify the correct rate. Refer to the iRMX 86 CONFIGURATION GUIDE for the procedures.

NUCLEUS DEMONSTRATION SYSTEM

LOADING THE NUCLEUS DEMONSTRATION SYSTEM

You can use either the ICE-86 In-Circuit Emulator or the iSBC 957A/B package to load the Nucleus demonstration system from diskette to memory. Using either product, load the following files from diskette to memory:

```
NUCLUS.DMO
DEBGGR.DMO
TBASIC.DMO
ROOTJB.DMO
```

Load file ROOTJB.DMO last because it contains instructions that initialize iAPX 86 registers to their proper values.

To load and start the Nucleus demonstration system with the ICE-86 Emulator, place a system diskette containing ICE-86 software in drive F0 of the Development System and the Nucleus demonstration diskette in drive F1. Enter the following commands at the keyboard of the Development System (the files are already located):

```
ICE86
LOAD :F1:NUCLUS.DMO
LOAD :F1:DEBGGR.DMO
LOAD :F1:TBASIC.DMO
LOAD :F1:ROOTJB.DMO
GO
```

Refer to the ICE-86 IN-CIRCUIT EMULATOR OPERATION INSTRUCTIONS FOR ISIS-II USERS for complete instructions on the use of the ICE-86 Emulator.

To load and start the Nucleus demonstration system with the iSBC 957A/B package, place a system diskette containing the iSBC 957A/B software in drive F0 of the Development System and the Nucleus demonstration diskette in drive F1. Enter the following commands at the keyboard of the Development System if you are using the iSBC 957B package:

```
APXL0D
L :F1:NUCLUS.DMO
L :F1:DEBGGR.DMO
L :F1:TBASIC.DMO
L :F1:ROOTJB.DMO
G
```

If you are using the iSBC 957A package, enter "SBC861" instead of "APXL0D". The remaining commands stay the same. Refer to the iSBC 957A INTELLEC -- iSBC 86/12A INTERFACE AND EXECUTION PACKAGE USER'S GUIDE or the USER'S GUIDE FOR THE iSBC 957B iAPX 86,88 INTERFACE AND EXECUTION PACKAGE for complete instructions on the use of the iSBC 957A/B package.

NUCLEUS DEMONSTRATION SYSTEM

A few seconds after you have entered the GO (or G) command at the Development System keyboard to start execution of the system, the following message appears on the terminal connected to the iSBC 86/12A computer, indicating that the TBASIC interpreter is ready for use:

```
iRMX 86 TINY BASIC DEMO. V2.0  
READY
```

USING THE NUCLEUS DEMONSTRATION SYSTEM

After you have initiated execution of the Nucleus demonstration system, the TBASIC interpreter displays a "greater than" (>) prompt at the terminal to indicate that it is ready for use. The characteristics of the TBASIC interpreter are similar to those of most BASIC interpreters. It allows you to enter and run a subset of BASIC language statements. It either interprets the statements as they are entered, or it stores the statements in memory and processes them as a complete program. The TBASIC interpreter also contains commands and functions that perform such iRMX 86 functions as creating tasks and sending messages. The following sections describe the operations of the TBASIC interpreter.

OPERATING MODES

The interpreter has two operating modes: interactive and deferred. In interactive mode you enter an individual statement followed by a carriage return. The interpreter processes the statement and then prompts you for a new statement line. In deferred mode, you create entire programs by entering a series of statements that are stored by the interpreter. Interpretation and execution of the stored statements is deferred until you enter a RUN statement.

You specify either the interactive or deferred mode by the absence or presence of a preceding line number for each statement line you enter.

For the interactive mode, enter a statement line without a preceding line number. (Some statements, such as RUN, LIST and NEW can only be entered in interactive mode.) For example, the statements:

```
RUN carriage return  
or  
NEW carriage return
```

would be interpreted in interactive mode. As soon as the interpreter executes either statement, it will prompt you for a new statement.

For deferred mode, enter a series of statement lines, each with a preceding line number, space once, and enter a statement line. Line numbers are normally entered in sequential increments of 10. For example, the numbered statement lines:

NUCLEUS DEMONSTRATION SYSTEM

```
NEW
10 REM                               THIS IS A DEFERRED MODE EXAMPLE
20 LET T = CRTSEGM (128,.S)
.
.
.
70 GOTO 40
.
.
.
RUN
```

would be stored as part of a program, and the interpretation and execution of all numbered statement lines would be deferred until the RUN statement was encountered. When you enter the RUN statement, the program statements are interpreted in sequential order (which is not necessarily the order in which you entered the statements).

CHANGING STATEMENT LINES

You can change a line of a program by entering a new line with the same line number. The interpreter disregards all but the last occurrence of a line with a given line number. You can delete a line by entering the line number alone, followed by a carriage return. The interpreter treats a line containing only a line number as a null operation.

VARIABLES

TBASIC supports two kinds of variables. A variable can be either a single alphabetic character (A-Z) or an array element. The interpreter does not distinguish between uppercase and lowercase characters. You can use one array only: the special character "@" followed by an index. The interpreter dynamically allocates space for this array. You can determine the amount of space available in the array by calling the SIZE function described later in this chapter.

Examples:

The following are acceptable variable names:

```
A
F
@(3)
@(expression) where expression is a BASIC expression as described
later in this chapter
```

NUCLEUS DEMONSTRATION SYSTEM

The following are not acceptable as variable names:

5
AB
2C
INTEG
F(2)

CONSTANTS

The interpreter supports integer constants in the range -32768 through +32767. It always interprets constants as decimal numbers.

EXPRESSIONS

Valid expressions can be built from the following:

- Integers (from -32768 to +32767)
- Variables (A-Z)
- Array elements (@(expression))
- Arithmetic operators (+, -, /, *)
- Relational operators (>, <, >=, <=, =, # (not equal to))

STATEMENTS AND FUNCTIONS

The TBASIC interpreter contains a number of statements and functions. Some of these are normally associated with any BASIC interpreter and others perform iRMX 86 Nucleus operations. The following sections describe these statements and functions. Table 4-1 provides a convenient dictionary that lists all of the TBASIC statements and functions in alphabetical order.

The following conventions are used in the descriptions of all statements and functions in this chapter.

- [parameter] The brackets are used to delimit optional parameters.
- |parameters| The vertical lines delimit a number of parameters separated with commas. You have the choice of entering any one, but only one, of the delimited parameters.

NOTE

When entering TBASIC and iRMX 86 functions, do not put any spaces between the delimiting parentheses.

NUCLEUS DEMONSTRATION SYSTEM

Table 4-1. Statement And Function Dictionary

Language Element	Definition	Page
ABS	Returns an absolute value.	4-11
CATALOG	Catalogs an object.	4-13
CRTMBOX	Creates a mailbox.	4-13
CRTSEGM	Creates a segment.	4-14
CRTSEMA	Creates a semaphore.	4-14
CRTTASK	Creates a task.	4-15
DELMBOX	Deletes a mailbox.	4-16
DELSEGM	Deletes a segment.	4-16
DELSEMA	Deletes a semaphore.	4-17
DELTASK	Deletes a task.	4-17
FOR	Starts a loop.	4-7
GETTKNS	Gets a token for an object.	4-17
GOSUB	Transfers control to a subroutine.	4-8
GOTO	Transfers control to a line.	4-8
IF	Processes a statement conditionally.	4-8
INPUT	Allows variable assignment from the console.	4-9
LET	Assigns a value to a variable.	4-9
LIST	Lists the current program.	4-10
LOOKUPO	Looks up a name in an object directory.	4-18
NEW	Clears memory of all source statements.	4-10
NEXT	Provides looping control.	4-7
PRINT	Prints a line at the console.	4-10
RCVUNIT	Receives units from a semaphore.	4-19
RECVMSG	Receives an object from a mailbox.	4-19

NUCLEUS DEMONSTRATION SYSTEM

Table 4-1. Statement And Function Dictionary (continued)

Language Element	Definition	Page
REM	Indicates a comment line.	4-11
RESTASK	Resumes a task.	4-20
RETURN	Returns control from a subroutine.	4-11
RND	Generates a random number.	4-12
RUN	Runs the stored program.	4-11
SENDMSG	Sends an object to a mailbox.	4-21
SIZE	Returns the size of the available array storage space.	4-12
SLEEP	Places the interpreter in the asleep state.	4-21
SNDUNIT	Sends units to a semaphore.	4-22
STOP	Stops program execution.	4-11
SUSTASK	Suspends a task.	4-22
UNCATLG	Deletes a name from an object directory.	4-23

BASIC STATEMENTS

This section describes the statements available with the TBASIC interpreter that are normally a part of any BASIC interpreter.

FOR ... NEXT

These statements provide looping control. The formats are as follows:

```
FOR var-name = start-val TO end-val [STEP inc-val]
```

```
NEXT var-name
```

where:

```
var-name      Variable used as a loop counter.
```

NUCLEUS DEMONSTRATION SYSTEM

`start-val` Starting value of the loop counter.

`end-val` Ending value of the loop counter.

`incr-val` Amount that the loop counter increments each time a loop begins. If STEP `inc-val` is not specified, the default value is 1.

The interpreter performs all statements delimited by the FOR and NEXT statements until the value of `var-name` is greater than `end-val`. You can use nested loops.

GOSUB

This statement transfers control to a subroutine. The format is as follows:

GOSUB `line-number`

where:

`line-number` Line number containing the first statement of the subroutine.

Subroutines may be recursive.

GOTO

This statement transfers control to another statement. The format is as follows:

GOTO `line-number`

where:

`line-number` Line number of the statement to which GOTO transfers control.

IF

This statement provides for conditional execution of a statement. The format is as follows:

IF `condition` `statement`

NUCLEUS DEMONSTRATION SYSTEM

where:

condition	Expression containing a relational operator. If condition is true, statement is executed; otherwise, control passes to the next line.
statement	A TBASIC statement that is executed only if condition is true.

INPUT

This statement halts a running program until you enter a numerical value through the keyboard. The value you enter is assigned to the variable in the statement, and the program resumes execution. The format is as follows:

```
INPUT var-name
```

where:

var-name	Variable name which is assigned a value from the console.
----------	---

When execution is halted, the interpreter displays the following prompt:

```
var-name :
```

and then waits for your input which it assigns to var-name.

LET

This statement assigns a value to a variable. The format is as follows:

```
[LET] var-name = expression
```

where:

var-name	Variable to which a value is assigned.
expression	Expression whose value is assigned to var-name.

The word LET is optional.

NUCLEUS DEMONSTRATION SYSTEM

LIST

This statement lists part or all of the program currently in memory. You can enter this statement in interactive mode only. The format is as follows:

```
LIST [line-number]
```

where:

line-number	Line number at which you want the listing to begin. LIST lists the remainder of the program. If you omit this parameter, the entire program is listed.
-------------	--

NEW

This statement clears memory of all stored statements. You can enter this statement in interactive mode only. The format is as follows:

```
NEW
```

Failure to clear memory before entering a new program in deferred mode causes unpredictable results when the new program is executed.

PRINT

This statement prints a line at the console. The format is as follows:

```
PRINT [field-width] >expression , "quoted-string"> [, . . .]
```

where:

field-width	Decimal value indicating the width of the field for numeric output. Output is right-justified in this field. If this value is not specified, a default field width of 6 is assumed.
expression	Any legitimate expression; the expression is evaluated before printing.
"quoted-string"	A string of characters enclosed in double quotes; the string is printed exactly as entered.
. . .	Indicates that a number of expressions and quoted strings, separated in the command by commas, can be printed on the same line.

If you do not specify any parameters, the interpreter prints a blank line.

NUCLEUS DEMONSTRATION SYSTEM

REM

This statement allows you to place non-executable remarks in your source code. The format is as follows:

REM comment

where:

comment Any comment you wish to place in your program list.

RETURN

This statement returns control from a subroutine. The format is as follows:

RETURN

RUN

This statement starts the execution of the program currently stored in memory. You can enter this statement in interactive mode only. Its format is as follows:

RUN

STOP

This statement stops the execution of the program. The format is as follows:

STOP

BASIC FUNCTIONS

This section describes the functions available with the TBASIC interpreter that are normally a part of any BASIC interpreter. The functions can be used anywhere that TBASIC expressions can be used.

ABS

This function returns the absolute value of an expression. The format is as follows:

ABS (expression)

NUCLEUS DEMONSTRATION SYSTEM

where:

expression Any valid TBASIC expression for which the absolute value is desired.

RND

This function returns a random number between 1 and the value of an expression. The format is as follows:

RND (expression)

where:

expression Any valid TBASIC expression. RND returns a random number between 1 and expression.

SIZE

This function returns the current size (in bytes) of the available array storage space. This value is twice the number of elements available in array @. Since the interpreter uses a fixed amount of memory to store both programs and data, a large program is allowed fewer array elements than a small program. The format is as follows:

SIZE

iRMX 86 STATEMENTS AND FUNCTIONS

This section describes the TBASIC statements and functions that allow you to make iRMX 86 system calls. The parameters for these statements and functions are very similar to the parameters for the equivalent Nucleus system calls. If you are unsure about the parameters for any of these BASIC statements and functions, refer to the iRMX 86 NUCLEUS REFERENCE MANUAL for complete descriptions.

Many of these statements and functions require you to supply a variable in which the interpreter returns the status of the iRMX 86 system call. Any variable you enter on the right-hand side of an equal sign (=) must be preceded by a period (.) if TBASIC is to return a value. You need not include the period when you later reference the variable.

The functions in this section are described as if they are part of assignment statements. However, they can be used anywhere that a TBASIC expression can be used.

NUCLEUS DEMONSTRATION SYSTEM

CATALOG

This statement catalogs a named object in a given object directory. The format is as follows:

```
CATALOG (job$token,object$token,"name",.stat$var)
```

where:

job\$token	Token for the job in whose object directory the object is to be cataloged. A zero value for this parameter indicates that the calling task's object directory is used.
object\$token	Token for the object to be cataloged.
"name"	One- to twelve-ASCII-character name under which the object is cataloged. This name must be enclosed in double quotes.
.stat\$var	Variable in which the interpreter returns the status of the catalog operation.

Example:

The following statement catalogs an object as MYTASK in the object directory of the calling task's job (the interpreter's job). Variable T contains a token for the object to be cataloged. The status of the catalog operation is returned in variable S:

```
CATALOG (0,T,"MYTASK",.S)
```

CRTMBOX

This function creates a mailbox and returns a token for it. The format is as follows:

```
mbox$token = CRTMBOX (mbox$flags,.stat$var)
```

where:

mbox\$token	Variable in which the interpreter returns a token for the newly created mailbox.
mbox\$flags	Value indicating how tasks are to be queued at the new mailbox. Possible values include: 0 First-in, first-out 1 Priority

NUCLEUS DEMONSTRATION SYSTEM

`.stat$var` Variable in which the interpreter returns the status of the create mailbox operation.

Example:

The following function creates a mailbox with priority-based queuing and returns a token for it in variable M. It returns the status of the create operation in variable S:

```
LET M = CRTMBOX (1,.S)
```

CRTSEGM

This function creates a segment and returns a token for it. The format is as follows.

```
seg$token = CRTSEGM (size,.stat$var)
```

where:

`seg$token` Variable in which the interpreter returns a token for the newly created segment.

`size` Size in bytes of the segment. A zero indicates that a 64K-segment is requested.

`.stat$var` Variable in which the interpreter returns the status of the create segment operation.

Example:

The following function call creates a 128-byte segment and returns a token for it in variable T. It returns the status of the create operation in variable S:

```
LET T = CRTSEGM (128,.S)
```

CRTSEMA

This function creates a semaphore. The format is as follows:

```
sema$token = CRTSEMA (init$value,max$value,sema$flags,.stat$var)
```

where:

`sema$token` Variable in which the interpreter returns the token of the newly-created semaphore.

`init$value` Initial value of the semaphore.

NUCLEUS DEMONSTRATION SYSTEM

max\$value	Maximum value of the semaphore.
sema\$flags	Value indicating how tasks are to be queued at the new semaphore. Possible values are: 0 First-in, first-out 1 Priority
.stat\$var	Variable in which the interpreter returns the status of the create semaphore operation.

Example:

The following function call creates a semaphore with priority-based queuing, with initial and maximum values of 1, and returns a token for it in variable T. It also returns the status of the create operation in variable S:

```
LET T = CRTSEMA (1,1,1,.S)
```

CRTTASK

This function creates a task and returns a token for that task. The format is as follows:

```
task$token = CRTTASK(pri,@start$addr,data$seg,@stack$ptr,  
                    stack$size,0,.stat$var)
```

where:

task\$token	Variable in which the interpreter returns a token for the created task.
pri	Priority of the task.
@start\$addr	Pointer indicating the task's starting address.
data\$seg	Base value of the task's data segment. A zero indicates that the task dynamically initializes the data segment register.
@stack\$ptr	Pointer indicating the address of the stack segment. A value of 0:0 indicates that the Nucleus assigns a stack segment when the task is created.
stack\$size	Size of the stack, in bytes.
.stat\$var	Variable in which the interpreter returns the status of the create task operation.

NUCLEUS DEMONSTRATION SYSTEM

Example:

The following function call creates a task with a priority of 129, a start address of 14A0:343 (obtained from the locate map for that task), and a stack size of 600. The task creates the data segment and the stack segment. The interpreter returns a token for the newly-created task in variable T, and the status of the operation in variable S:

```
LET T = CRTTASK (129,@14A0:343,0,@0:0,600,0,.S)
```

DELMBOX

This statement deletes a mailbox. The format is as follows:

```
DELMBOX (mbox$token,.stat$var)
```

where:

mbox\$token	Token of the mailbox to be deleted.
.stat\$var	Variable in which the interpreter returns the status of the delete mailbox operation.

Example:

The following statement deletes a mailbox and returns the status of the delete operation in variable S. Variable T contains a token for the mailbox:

```
DELMBOX (T,.S)
```

DELSEGM

This statement deletes a segment. The format is as follows:

```
DELSEGM (seg$token,.stat$var)
```

where:

seg\$token	Token for the segment being deleted.
.stat\$var	Variable in which the interpreter returns the status of the delete segment operation.

Example:

The following statement deletes a segment and returns the status in variable S. Variable T contains a token for the segment:

```
DELSEGM (T,.S)
```

NUCLEUS DEMONSTRATION SYSTEM

DELSEMA

This statement deletes a semaphore. The format is as follows:

```
DELSEMA (sema$token,.stat$var)
```

where:

sema\$token	Token for the semaphore to be deleted.
.stat\$var	Variable in which the interpreter returns the status of the delete semaphore operation.

Example:

The following statement deletes a semaphore and returns the status in variable S. Variable T contains a token for the semaphore:

```
DELSEMA (T,.S)
```

DELTASK

This statement deletes a task. The format is as follows:

```
DELTASK (task$token,.stat$var)
```

where:

task\$token	Token of the task to be deleted.
stat\$var	Variable in which the interpreter returns the status of the delete task operation.

Example:

The following statement deletes a task and returns the status of the delete operation in variable S. Variable T contains a token for the task to be deleted:

```
DELTASK (T,.S)
```

GETTKNS

This function returns a token for an object. The format is as follows:

```
obj$token = GETTKNS (select$val,.stat$var)
```

NUCLEUS DEMONSTRATION SYSTEM

where:

obj\$token	The requested token.
select\$val	Value indicating the object for which a token is requested. Possible values are: 0 Token for the interpreter task. 1 Token for the interpreter task's job. 2 Token for the interpreter job's parameter object. 3 Token for the root job.
.stat\$var	Variable in which the interpreter returns the status of the operation.

Example:

The following function call returns a token for the root job in variable T and returns the status of the operation in variable S:

```
LET T = GETTKNS (3,.S)
```

LOOKUPO

This function looks up a name in an object directory and returns a token for that object. The format is as follows:

```
obj$token = LOOKUPO (job$token,"name",time$limit,.stat$var)
```

where:

obj\$token	Variable in which the interpreter returns a token for the object.
job\$token	Token for the job in whose object directory the function searches for the name. A zero indicates that the interpreter's job is searched.
"name"	Name under which the object is cataloged. This name must be enclosed in double quotes.
time\$limit	Number of system time units that the function is willing to wait for the name to become available.
.stat\$var	Variable in which the interpreter returns the status of the lookup operation.

NUCLEUS DEMONSTRATION SYSTEM

Example:

The following function call looks up the name MYTASK in the interpreter's object directory, does not wait if the name is not there, and, if the desired entry is found, returns a token for the object in variable T. It returns the status of the operation in variable S:

```
LET T = LOOKUPO (0,"MYTASK",0,.S)
```

RCVUNIT

This function receives units from a semaphore and returns the new value of the semaphore. The format is as follows:

```
value = RCVUNIT (sema$token,units,time$limit,.stat$var)
```

where:

value	Variable in which the interpreter returns the number of units remaining in the custody of the semaphore after the units have been received.
sema\$token	Token for the semaphore.
units	Number of units to be received from the semaphore.
time\$limit	Number of system time units the interpreter is to wait for the units.
.stat\$var	Variable in which the interpreter returns the status of the receive units operation.

Example:

The following function call receives one unit from a semaphore and does not wait for the unit to become available. It returns the new value of the semaphore in variable V and the status of the receive operation in variable S. Variable T contains a token for the semaphore:

```
LET V = RCVUNIT (T,1,0,.S)
```

RCVMSG

This function waits for an object at a mailbox and returns a token for the object if one is available. The format is as follows:

```
mess$token = RCVMSG (mbox$token,time$limit,.resp$var,.stat$var)
```

NUCLEUS DEMONSTRATION SYSTEM

where:

mess\$token	Variable in which the interpreter returns a token for the object.
mbox\$token	Token for the mailbox.
time\$limit	Number of system time units the interpreter is to wait for the object.
.resp\$var	Variable in which the interpreter returns a token for the response mailbox or semaphore, if a response is requested.
.stat\$var	Variable in which the interpreter returns the status of the receive message operation.

Example:

The following function call receives an object from a mailbox without waiting, returns a token for the object in variable T, returns a token for the response mailbox in variable R, and returns the status of the receive operation in variable S:

```
LET T = RECVMSG (M,0,.R,.S)
```

RESTASK

This statement resumes a suspended task. The format is as follows:

```
RESTASK (task$token,.stat$var)
```

where:

task\$token	Token of the task to be resumed.
.stat\$var	Variable in which the interpreter returns the status of the resume operation.

Example:

The following statement resumes a suspended task and returns the status of the resume operation in variable S. Variable T contains a token for the suspended task:

```
RESTASK (T,.S)
```

NUCLEUS DEMONSTRATION SYSTEM

SENDMSG

This statement sends a message (in the form of an object) to a mailbox. The format is as follows:

```
SENDMSG (mbox$token,obj$token,response$token,.stat$var)
```

where:

mbox\$token	Token of the mailbox to which the message is being sent.
obj\$token	Token of the object being sent.
response\$token	Token of the desired response mailbox or semaphore. A zero indicates that no response is desired.
.stat\$var	Variable in which the interpreter returns the status of the send message operation.

Example:

The following statement sends an object to a mailbox and specifies a response mailbox at which the receiving task can acknowledge receiving the object. Variable M contains a mailbox token, variable A contains an object token, and variable R contains a response mailbox token. The interpreter returns status in variable S:

```
SENDMSG (M,A,R,.S)
```

SLEEP

This statement places the calling task (the TBASIC interpreter) in the asleep state. The format is as follows:

```
SLEEP (units,.stat$var)
```

where:

units	Number of system time units that the interpreter task is willing to sleep. A zero value places the task on the ready task queue.
.stat\$var	Variable in which the interpreter returns the status of the sleep operation.

Example:

The following statement places the interpreter in the asleep state for one second and returns status of the sleep operation in variable S:

```
SLEEP (100,.S)
```

NUCLEUS DEMONSTRATION SYSTEM

SNDUNIT

This statement sends units to a semaphore. The format is as follows:

```
SNDUNIT (sema$token,num$units,.stat$var)
```

where:

sema\$token	Token of the semaphore to which units are being sent.
num\$units	Number of units to be sent to the semaphore.
.stat\$var	Variable in which the interpreter returns the status of the send units operation.

Example:

The following statement sends one unit to a semaphore and returns the status of the operation in variable S. Variable T contains a token for the semaphore:

```
SNDUNIT (T,1,.S)
```

SUSTASK

This statement suspends a task. The format is as follows:

```
SUSTASK (task$token,.stat$var)
```

where:

task\$token	Token of the task to be suspended.
.stat\$var	Variable in which the interpreter returns the status of the suspend operation.

Example:

The following statement suspends a task and returns the status of the suspend operation in variable S. Variable T contains a token for the task to be suspended:

```
SUSTASK (T,.S)
```

NUCLEUS DEMONSTRATION SYSTEM

UNCATLG

This statement deletes an entry from an object directory. The format of this statement is as follows:

```
UNCATLG (job$token, "name", .stat$var)
```

where:

job\$token	Token for the job from whose object directory the name is to be deleted. A zero indicates the object directory of the job containing the calling task.
"name"	Name to be deleted from the object directory. This name must be enclosed in double quotes.
.stat\$var	Variable in which the interpreter returns the status of the uncatalog operation.

Example:

The following statement removes the entry with the name MYTASK from the interpreter's object directory and returns the status of the operation in variable S:

```
UNCATLG (0, "MYTASK", .S)
```



CHAPTER 5. START-UP SYSTEM

The Start-Up System is provided for two purposes. First, it allows you to become familiar with some of the capabilities of the Human Interface, a major iRMX 86 subsystem. Second, it enables you to manipulate files on iRMX 86 diskettes; such manipulation is not possible on an INTELLEC Development System.

Included in your iRMX 86 package is another facility for manipulating iRMX 86 files on secondary storage devices. It is the Files Utility, which is described in Chapter 8 of this manual. Generally, the Start-Up System is the preferred file manipulator, because of the following reasons:

- For users who plan to use the Human Interface subsystem, it is desirable to gain experience with some Human Interface capabilities.
- The Start-Up System has a facility that allows users to add commands to the system's command set.
- Because its commands are not resident in memory, the Start-Up System's memory requirements are not proportional to the number of commands it offers.

The principal reason for choosing the Files Utility over the Start-Up System is that the latter is initially configured for use with only 8-inch, single-density diskettes. The Files Utility imposes no such medium constraint.

FUNCTIONS PROVIDED

The Start-Up System is an iRMX 86 application system that allows you to perform the following operations:

- Format iRMX 86 diskettes.
- Copy files from ISIS-II diskettes to iRMX 86 diskettes.
- Copy files from iRMX 86 diskettes to ISIS-II diskettes.
- Copy files between iRMX 86 diskettes.
- Delete files from iRMX 86 diskettes.
- Create directories on iRMX 86 diskettes.
- Display, on the application system terminal, the contents of directories on iRMX 86 diskettes.

START-UP SYSTEM

HARDWARE REQUIRED

The Start-Up System requires the following hardware:

- An INTELLEC Development System with at least 64K bytes of memory and at least one flexible disk drive and one other drive.
- An iSBC 86/12A Single Board Computer with at least 192K of contiguous random access memory (RAM) starting at address 0. For certain operations, more than 192K of RAM is required; for example, if you plan to execute nested SUBMIT files, you will need more than 192K.
- An iSBC 957A/B INTELLEC --- Interface and Execution Package. The cable in the package should be installed for parallel interfacing between the iSBC 86/12A board and the INTELLEC Development System. The PROMs in the package should be installed in the iSBC 86/12A board.
- An iSBC 204, iSBC 208, or iSBC 215/218 controller set for 8-inch drives and 128 bytes per sector.
- At least one 8-inch diskette drive.
- A cable for connecting the controller to the diskette device.
- An iRMX 86 diskette containing the Start-Up System.
- An ISIS-II Utilities diskette containing the Start-Up System.
- A terminal set for 9600 baud and no parity checking.
- A cable for connecting the terminal to the serial port of the iSBC 86/12A board.

USING THE START-UP SYSTEM

The following sections describe how to initiate the Start-Up System, list the available commands, and enumerate some special use restrictions that do not normally apply to the Human Interface.

GETTING STARTED

After assembling your hardware as indicated in Chapter 3 and turning on the power, perform the following steps:

1. Look at the display of your application terminal. If asterisks (*) are gradually filling the screen, your system is ready for use. Otherwise, reset the system. This should start your screen to fill with asterisks. If this doesn't work, you have a hardware problem that must be fixed before you can continue.

START-UP SYSTEM

2. Insert an ISIS-II system diskette with the iSBC 957B software in drive 0 of your INTELLEC Development System. Insert the ISIS-II Utilities diskette in drive 1 of your development system. Insert the iRMX 86 diskette containing the Start-Up System in drive 0 of your application system hardware. If applicable, insert another diskette in drive 1 of your application system hardware.
3. At your development system terminal, enter

```
APXLOD <cr>
```

This causes the following display at the development system terminal:

```
ISIS-II iAPX 86, 88 LOADER Vx.x
```

```
iAPX 86, 88 Monitor, Vx.x
```

followed by the monitor prompt, a period (.).

4. At your development system terminal, enter the letter r, followed by the name :F1:BS1 and <cr>. This bootstraps the Start-Up System into the application system memory.
5. When an asterisk (*) appears, enter <cr>. This initiates the Start-Up System. After a short wait, the following display will appear on the application system terminal:

```
iRMX 86 HUMAN INTERFACE, Vx.x: user = WORLD
```

followed by the Human Interface prompt, a hyphen (-). You are now ready to begin entering Human Interface commands.

An alternative to steps 3, 4, and 5 is to enter, at the development system, the ISIS-II command

```
SUBMIT :F1:SUPLD(:F1:)
```

This causes the following display at the development system terminal:

```
apxlod  
ISIS-II iAPX 86, 88 LOADER, Vx.x  
iAPX 86, 88 MONITOR Vx.x
```

```
.l :F1:nuclus  
.l :F1:ios  
.l :F1:eios  
.l :F1:loader  
.l :F1:hi  
.l :F1:sroot  
.e
```

followed by the ISIS prompt, a hyphen (-).

START-UP SYSTEM

At your development system terminal, enter

APXLOD <cr>

and the letter g to initialize the Start-Up System.

START-UP SYSTEM COMMANDS

The commands that are supplied as part of the Start-Up System are described here in very general terms. For details concerning the effects of these commands, and their syntax requirements, refer to the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL.

File Management Commands

The following commands are available for managing files and related activities.

ATTACHDEVICE	Attaches an application system drive, which then can be addressed for purposes of file manipulation. Drives F0, F1, WF0, and WF1 need not be attached, because this is done automatically during initialization of the Start-Up System. Refer to Table 3-1 of this manual when using this command.
CREATEDIR	Creates one or more new directories.
COPY	Creates new data files, or copies files to new filenames.
DELETE	Deletes data files or empty directories.
DETACHDEVICE	Detaches an application system drive. The drive must be reattached before it can be used for file manipulation.
DIR	Lists a directory's file names (and, optionally, file attributes).
DOWNCOPY	Copies files and directories from an iRMX 86 volume to an ISIS-II secondary storage volume.
FORMAT	Formats an iRMX 86 diskette.
RENAME	Renames files or directories.
UPCOPY	Copies files and directories from an ISIS-II secondary storage volume to an iRMX 86 volume.

START-UP SYSTEM

Additional Services

The following commands provide additional services:

DATE	Sets or resets the system date, or displays the current date.
DEBUG	Transfers control to the iSBC 957A/B package, usually for the purpose of debugging an iRMX 86 application program.
SUBMIT	Loads and executes a sequence of commands from an iRMX 86 file on a secondary storage device.
TIME	Sets or resets the system clock, or displays the current system time.

The following SUBMIT files also provide additional services:

SGEN.CSD	Allows you to customize your own Start-up System.
SUPLD.CSD	Uploads the Start-up System into the application system memory.
SMAKE.CSD	Makes an iRMX 86 Start-up System diskette by taking the following steps: <ol style="list-style-type: none">1. Formats the iRMX 86 diskette in drive F0.2. Creates the SYSTEM directory on drive F0.3. Upcopies the Start-up System copy submit file SCOPY.CSD and changes its name to STARTUPCOPY.4. Upcopies all of the system commands and the Bootstrap Load file.
STARTUPCOPY	Formats an iRMX 86 diskette and copies all system start-up files to this diskette from another iRMX 86 diskette.

SPECIAL USE RESTRICTIONS

The following paragraphs describe some use restrictions that are not generally true of application systems that include the Human Interface.

The logical names :SYSTEM: and :PROG:, which, in the Human Interface subsystem, refer respectively to the system's collection of commands and your collection of special-purpose commands, are not available in the Start-Up System. However, you can achieve the effects of using :SYSTEM: and :PROG:, as they are described in the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL, by using SYSTEM/ and PROG/ instead of :SYSTEM: and :PROG:, respectively.

START-UP SYSTEM

If your application hardware has iSBC 215/218 controllers, each file name (including command names, because each command is a file) must be preceded by the logical name :WFn:, where n is 0 or 1, depending upon which drive contains the file. If other controllers are used instead, use the logical name that was specified when the ATTACHDEVICE command was entered.

CHAPTER 6. PATCHING UTILITY

The 8086 Patching Utility provides you with a method of modifying relocatable object modules with software updates or repair code. The process requires that the replacement code must first be generated with the ASM86 Assembler.

You can update a module to a newer version or add repair code in two ways:

- As a patch that generates a jump instruction to the replacement code and appends the replacement code to the end of the original module.
- As an in-place patch that directly overlays the replacement code on that of the original module.

An example of each technique is provided later in this chapter.

The replacement code itself may be supplied by Intel in either of two forms:

- An Intel-supplied object file, on diskette. In this case, all of the coding and assembly has been done; you need only invoke the Patching Utility to effect the replacement.
- An Intel-supplied source code listing with instructions for inserting the replacement code; in this case, much of the preliminary work has been done and you need little or no knowledge of ASM86 Macro Assembly Language to generate the replacement object module.

You can use the Patching Utility to modify your application modules by creating your own replacement module; in this case, a working knowledge of ASM86 Macro Assembly Language is required.

Version 1.0 of the Patching Utility is designed to run on an INTELLEC Series II or Series III Microcomputer Development System and version 2.0 of the Patching Utility is designed to run on an iRMX 86 system with at least one disk drive.

A patched module retains translator header record names for the original module, plus the names for the patch modules. By using either version of the Patching Utility, you can display the complete list of names to determine a module's update status. By using the iRMX 86 version of the Patching Utility you can also display translator header record names for all modules in a library. This allows you to determine the update status of a library.

PATCHING UTILITY

INVOKING THE PATCHING UTILITY

You prepare to invoke the Patching Utility by placing a diskette containing the INTELLEC version of the utility in drive 0 of your INTELLEC Development System. If you are using the iRMX 86 version of the Patch Utility, then you can invoke the utility by upcopying the Patch Utility file (PTCH86.R86) from diskette to an iRMX 86 secondary storage peripheral. In either case, the next step is to call the Patching Utility by entering the PTCH86 command in the form:

```
PTCH86 filename [segmentname segmentattribute]
```

where:

filename	Name of file containing an 8086 object module produced by PL/M-86, ASM86, or LINK86.
segmentname	Name of the segment whose combined-type parameter is to be modified. The name must be a valid segment name. The attributes listed between the brackets are optional. If segmentname is used, then segmentattribute must be used also.
segmentattribute	Keyword switch that determines the combine-type attribute given to the named segment. You must specify the attribute as either COMMON or PUBLIC. The COMMON attribute allows patch code to be overlaid on the segment. The PUBLIC attribute returns the segment to the combination mode normally given by the PL/M-86 compiler.

See the 8086/8087/8088 MACRO ASSEMBLY LANGUAGE REFERENCE MANUAL FOR 8086-BASED DEVELOPMENT SYSTEMS for a more detailed explanation of combine-type segments.

The INTELLEC version of the Patching Utility responds by displaying

```
8086 OBJECT PATCHING UTILITY, V1.0
```

on the INTELLEC console. The iRMX 86 version of the Patching Utility responds by displaying

```
iRMX 86 8086 OBJECT PATCH UTILITY, V2.0
```

on a console under the control of the iRMX 86 Human Interface. In either case, this display is followed by an indication of the outcome of the patch operation and then the Operating System prompt. If the invocation line contained the optional segmentname and segmentattribute, the message "ATTRIBUTE MODIFIED" is displayed. If the invocation line contained only the filename, the translator header records within the named object file or library are displayed. These translator header records serve to identify the patches which have been made to a module (described later in this chapter). Otherwise, the patch operation failed and an

PATCHING UTILITY

error message is displayed. Listing of translator header records in a library is supported only on the iRMX 86 version of the Patching Utility.

PATCHING PROCEDURES

Repair modules that you insert into existing modules must be generated with the ASM86 Assembler. To patch an independent object module containing errors (patching library modules is described later in this chapter), you invoke the Patching Utility to modify the combine-type attribute in the desired module segment to COMMON. This step allows you to use LINK86 to overlay the repair module on the segment to be patched. After linking with the repair module, you then use the Patching Utility to restore the PUBLIC attribute to the segment. The following example illustrates the steps for repairing independent object module files:

1. Enter the PTCH86 command to set the CODE segment combine-type attribute to COMMON, for example:

```
PTCH86 badmodule CODE COMMON
```

2. Enter the LINK86 command to overlay the repair object module on the original version, for example:

```
LINK86 badmodule, repairmodule TO newmodule
```

3. Enter the PTCH86 command to restore the CODE segment to PUBLIC, for example:

```
PTCH86 newmodule CODE PUBLIC
```

Typical examples of jump instruction overlays, in-place patch overlays, library module patching, and listing module header records are given in the following sections.

JUMP INSTRUCTION PATCH

In the following example, the module generates a patch that overlays a jump instruction on offset 0100H through 0102H of the original module. The jump transfers control to repair code at offset 0500H. The repair code is appended to the end of the module and is thus appended to that module.

PATCHING UTILITY

EXAMPLE:

```

NAME      REPAIR_V00001          ; Identifying module name.

CODE      SEGMENT      WORD COMMON      'CODE'
CGROUP    GROUP        CODE
          ASSUME       CS : CGROUP

          ORG          0100H      ; Offset of area in original
                                   ; module to be patched.

          JMP          REPAIRCODE

RETURN    LABEL        NEAR          ; Return here from repair area.

          ORG          0500H      ; Offset of end of original
                                   ; module.

REPAIRCODE:
;
;   (Repair goes here)
;
          JMP          RETURN      ; Return control to original
                                   ; module.

CODE      ENDS
END

```

IN-PLACE PATCH

The following example generates an in-place patch that directly overlays repair code on a module's previous code.

EXAMPLE:

```

NAME      REPAIR_V00002          ; Module name identification.

CODE      SEGMENT      WORD COMMON      'CODE'
CGROUP    GROUP        CODE
          ASSUME       CS : CGROUP

          ORG          0200H      ; Offset of the original
                                   ; operand.

          ADD          AX, 3      ; Replaces the original value
                                   ; with a "3"
                                   ; (the new instruction must be
                                   ; the same
                                   ; size as the original
                                   ; instruction).

CODE      ENDS

END

```


PATCHING UTILITY

PATCHING LIBRARY MODULES

To patch an object module that is located in a library, use the SUBMIT file (PATCH.CSD) supplied on your Utilities diskette. When invoked, the SUBMIT file will perform the following steps:

1. Enters a LINK86 command to separate the module to be patched from the library and put it in a temporary file.
2. Enters the PTCH86 command to set the CODE segment combine-type attribute to COMMON.
3. Enters a LINK86 command to overlay the replacement object module on the original version.
4. Enters the PTCH86 command to restore the CODE segment PUBLIC attribute.
5. Enters a LIB86 command to replace the original module in the library with the updated version.
6. Deletes the temporary files when the replacement is completed.

To invoke the SUBMIT file, enter the command in the following format. Note that the parentheses enclosing the parameter string and the embedded commas are required; embedded blanks are optional:

```
SUBMIT PATCH(library, oldmodule, segment, newmodule)
```

where:

library	Name of library containing the old module to be replaced.
oldmodule	Name of the module to be replaced.
segment	Name of the segment whose combine-type attribute is to be set to COMMON.
newmodule	Name of the file containing the replacement module code.

PATCHING UTILITY

LISTING TRANSLATOR HEADER RECORDS

If you want the Patching Utility to list an object module's translator header records on the console screen, enter the PTCH86 command without specifying the segment name or segment attribute. The listed records allow you to identify the patches that have been made to the module. A typical PTCH86 command entry and resulting header record display is as follows:

```
PTCH86 FILE.OBJ
ORIGINALMODULE
ORIGINALMODULE_REPAIR_V030-01
ORIGINALMODULE_REPAIR_V030-02
```

The "030" stands for version 3.0 of the software being patched, and "01" and "02" are the patch numbers of the Intel-supplied patches that have been made to the module. The iRMX 86 version of the Patching Utility can perform this listing operation on both object modules and libraries.

ERROR MESSAGES

When the Patching Utility encounters an error condition during a module repair session, it displays one of the following error messages:

ERROR nnn

A system call returned a non-zero error status, given as nnn. See the ISIS-II USER'S GUIDE for an explanation of numbered error messages in this form.

If you are using the iRMX 86 version of the Patch Utility, this type of error would appear as:

nnnn exception code mnemonic

See Table B-1 in this manual for a summary of these numbered error messages. See the appropriate iRMX 86 manual for detailed descriptions of iRMX 86 exception codes.

INVALID MODULE TYPE

The file specified is not suitable for the segment attribute modification and subsequent patching.

PATCHING UTILITY

INVALID RECORD TYPE

The object file contains an invalid record type for the object module format. Perhaps the wrong filename was entered, or the file contains code other than object code.

INVALID SYNTAX

The command line contains an error that was caused by a missing filename or a missing or misspelled keyword.

SEGMENT NOT FOUND

The desired record was not found before the end of the module.

CHAPTER 7. iRMX™ 86 DEVELOPMENT PROCEDURES

In order to produce a final iRMX 86-based application system for your users, you must go through two phases: a development phase and a production phase. During the development phase you design, build, and debug your system. In the production phase you produce the final systems for your users. This chapter outlines the steps you need to follow as you develop your iRMX 86-based application system. The steps illustrate the main points of the development process.

1. Define your application.
2. Do the high-level design, as follows:
 - Identify your hardware requirements.
 - Determine which of the iRMX 86 subsystems you need. The configurable nature of the iRMX 86 software allows you to select the parts that your application requires. It is recommended that you include the Debugger in your application system until it is fully developed. When you have completed the development process, you can remove the Debugger from your system to reduce memory requirements.
 - Divide your application into jobs and tasks. Assign task priorities, identify exchanges used for intertask communication, and determine the methods of interrupt handling and exception processing. Refer to the INTRODUCTION TO THE iRMX 86 OPERATING SYSTEM, iRMX 86 NUCLEUS REFERENCE MANUAL, iRMX 86 TERMINAL HANDLER REFERENCE MANUAL, and iRMX 86 DEBUGGER REFERENCE MANUAL for more information about these processes.
3. Write and debug the task code. As you finish writing each task, you can use either the ICE-86 In-Circuit Emulator or the iSBC 957A/B package to debug the task independently. Later, you can use the Debugger to debug the entire application.
4. Configure the application system. Do this by creating a system configuration file and an individual configuration file for each part of the Operating System. Assemble and compile all of the code, link it in the correct manner, and locate it at the proper addresses. See the iRMX 86 CONFIGURATION GUIDE for a detailed description of this process.
5. Assemble your hardware for testing the system.
6. If you are using the I/O Systems in your application, load the Start-Up System or the Files Utility System, format your iRMX 86 disks, and copy any necessary information to them.

IRMX™ 86 DEVELOPMENT PROCEDURES

7. Load your code into memory using one of the following:
 - The ICE-86 In-Circuit Emulator
 - The iSBC 957A INTELLEC -- iSBC 86/12A Interface and Execution Package
 - The iSBC 957B INTELLEC -- iAPX 86,88 Interface and Execution Package
 - The Bootstrap Loader
 - The Application Loader
 - The Human Interface (which calls the Application Loader)
8. Test and debug your system using the Debugger and either the ICE-86 In-Circuit Emulator or the iSBC 957A/B Interface and Execution package. Continue performing steps 3, 4, and 7 until you are satisfied with your system.
9. Unless you want the Debugger to be a permanent part of your system, perform step 4 again, but omit the Debugger.
10. Burn your debugged code into PROM and place it on your iAPX 86-based microcomputer system, or place your debugged code on an iRMX formatted diskette and use the Bootstrap Loader to load the code directly into memory.

Note that you can use the Bootstrap Loader to load your code at any stage of the development procedures, including the debugging stage (see the iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL for information on using the Bootstrap Loader).

CHAPTER 8. FILES UTILITY SYSTEM

The INTELLEC Microcomputer Development System does not recognize iRMX 86 diskette files. Consequently you cannot read, write, or format iRMX 86 diskettes directly from the ISIS-II operating system. However, you can perform these operations indirectly from the Development System by using the iRMX 86 Files Utility System.

FUNCTIONS PROVIDED

The iRMX 86 Files Utility System is an iRMX 86 application system that allows you to perform the following operations:

- Format an iRMX 86 diskette.
- Copy a file from an ISIS-II diskette to an iRMX 86 diskette.
- Copy a file from an iRMX 86 diskette to an ISIS-II diskette.
- Delete a file from an iRMX 86 diskette.
- Create a directory on an iRMX 86 diskette.
- Display, on the Development System terminal, the contents of a directory of an iRMX 86 diskette.

HARDWARE REQUIRED

The Files Utility System requires the following hardware:

- A Microcomputer Development System having at least 64k bytes of memory and at least one disk drive (hard or flexible).
- An iSBC 86/12A Single Board Computer with at least 192k bytes of memory and at least one disk drive (hard or flexible).
- The iSBC 957A INTELLEC -- iSBC 86/12A Interface and Execution Package or the iSBC 957B INTELLEC -- iAPX 86,88 Interface and Execution Package.

FILES UTILITY SYSTEM

STARTING THE FILES UTILITY

Before you can enter commands to the Files Utility, you must start it up. This involves connecting certain hardware modules and then entering appropriate commands on the INTELLEC Microcomputer Development System terminal.

After you have assembled your hardware, perform the following steps:

1. Place an ISIS-II system diskette containing the iSBC 957A/B software into drive 0 of your INTELLEC Microcomputer Development System and the Utilities diskette into any other drive.
2. Load the ISIS-II system.
3. Enter the following ISIS-II command:

```
SUBMIT :fx:FILES (:fx:)
```

where:

fx Identifier of the diskette drive containing the
Files Utility diskette.

When you enter this command, the ISIS-II operating system reads and processes the commands contained on the FILES.CSD file. These commands instruct the iSBC 957A/B monitor to load the Files Utility System from a diskette on the INTELLEC system into RAM on the iSBC 86/12A board.

After the ISIS-II system finishes processing the commands in the submit file, the system prompts for another command. Respond by entering

```
SBC861
```

This command instructs the ISIS-II system to connect you to the iSBC 957A monitor. If you are using the iSBC 957B package, the following command instructs the ISIS-II system to connect you to the iSBC 957B monitor.

```
APXLOD
```

In either case, the monitor signals you that it is ready to accept your next command by displaying a period (.) on the screen of your INTELLEC system. When the period appears, enter

```
G
```

This causes the Disk Utility System to begin running. The screen of your INTELLEC system should display the heading

```
iRMX 86 FILES UTILITY Vx.x
```

The Files Utility signals that it is ready to accept your next command by displaying an asterisk (*) on the screen of the INTELLEC system.

FILES UTILITY SYSTEM

USING THE FILES UTILITY

The Files Utility provides 10 file management commands, as follows:

ATTACHDEV	DIR
BREAK	DOWNCOPY
CREATEDIR	FORMAT
DELETE	HELP
DETACH	UPCOPY

The commands are described in alphabetical sequence later in this chapter. However, before actually using the commands, you should understand the diskette handling procedures and how the Files Utility System handles errors.

CHANGING DISKETTES

When the Files Utility is running and you have already performed an operation on a particular diskette, you cannot simply remove that diskette from the drive and replace it with another. The Utility System is not aware of diskette changes and treats the second diskette as if it were the first, and thereby possibly writes over or destroys valuable information. To change diskettes in a drive, you must enter a DETACH command to logically detach the drive from the system, change diskettes, and then (with one exception) enter an ATTACHDEV command to again logically attach the device.

The one exception to this command entry sequence is the FORMAT command. As described later in this chapter, this command writes iRMX 86 formatting information on blank diskettes. Since the FORMAT command always expects a blank diskette and a detached drive, you can replace diskettes in a drive any number of times if you use only the FORMAT command before entering the ATTACHDEV command. The FORMAT command will destroy the information, if any, previously contained on the diskette.

COMMANDS

This section provides descriptions of the Files Utility commands and their parameters in alphabetical sequence. Each command has a two-character abbreviation. You can use either the full name or its abbreviation when entering a command.

ATTACHDEV (AD)

This command attaches a physical device to the system and associates a logical name with the device. The command can also be used to display the current attachment of a logical name. The format is as follows:

FILES UTILITY SYSTEM

AD :logicalname:[= physicalname]

where:

:logicalname: A 1-to 12-character ASCII name, surrounded by colons.

= If used, there must be no spaces surrounding the equal sign.

physicalname Physical device name as configured in the I/O System (see Table 5-1). If physical name is omitted, the current attachment is displayed by default; for example:

AD :FO: (command entry)
:FO: = FX0 (displayed output)

BREAK (BR)

This command causes an exit from the Files Utility System to the iSBC 957A/B monitor. The format is as follows:

BR

CREATEDIR (CD)

This command creates an iRMX 86 directory file. The format is as follows:

CD rmx-pathname

where:

rmx-pathname Path name of the iRMX 86 directory file to be created.

DELETE (DE)

This command removes the specified iRMX 86 file from the directory where it is listed. The format command is as follows:

DE rmx-pathname

where:

rmx-pathname Path name of the iRMX 86 file to be deleted.

FILES UTILITY SYSTEM

DETACH (DT)

This command detaches a logical name from the system. The command is used for changing diskettes, prior to entering a FORMAT command, or to reconfigure a device to a different sector size. The format is as follows:

```
DT :logical-devicename:
```

where:

```
:logical-devicename: The logical name you assigned to a physical
device via an ATTACHDEV command.
```

DIR (DI)

This command lists an iRMX 86 directory file at the Development System console. The format is as follows:

```
DI rmx-pathname [S]
```

where:

```
rmx-pathname Path name of the iRMX 86 directory file to be
listed.
```

```
S Switch that causes a "long" or expanded display of
directory file that includes: file type (a "DR"
heading for a directory file or a blank heading for
a data file), number of blocks, and number of bytes
in file. If S is not specified, a "fast" format
will be displayed, consisting of file names only
```

The directory file listing includes a line that lists the size of the directory. This line appears as:

```
n FILES
```

In this line, n specifies the number of entries currently present in the directory.

DOWNCOPY (DC)

This command creates an ISIS-II file and copies the specified iRMX 86 file to it. If the ISIS-II file already exists, it is written over. The format is as follows:

```
DC rmx-pathname TO isis filename
```

FILES UTILITY SYSTEM

where:

`rmx-pathname` Path name of the iRMX 86 file to be copied.
`isis-filename` Name of the ISIS-II file to be created.

FORMAT (FO)

This command writes iRMX 86 formatting information on a diskette. All information previously contained on the diskette will be destroyed by the formatting operation. Each diskette must be formatted before it can be used by the iRMX 86 Operating System.

The FORMAT command expects an unattached drive. The drive device can either be unattached at system start up, or you can detach it by entering a DETACH command prior to entering the FORMAT command. Since the device remains unattached after FORMAT completes execution, you must attach the device by entering an ATTACHDEV command before entering any other Utility command except another FORMAT command. (See also the "Changing Diskettes" section in this chapter, and the ATTACHDEV and DETACH command descriptions.)

The FORMAT command contains parameters that are specified in the form "keyword=value". There must not be any spaces surrounding the equal sign. Also, you can abbreviate each of these keywords as shown. The abbreviations and the format of this command are as follows (brackets [] indicate optional parameters):

```
FO physicalname volumename [GRANULARITY=gran]
    [INTERLEAVE=ileave][NUMBERFNODES=nodes] [switch]
```

or

```
FO physicalname volumename [GR=gran] [IL=ileave]
    [NF=nodes] [switch]
```

where:

`physicalname` Physical device name for the drive, as configured in the I/O System, that denotes the iRMX 86 drive on which the diskette resides. Possible values are itemized in Table 3-1.

`volumename` A 1- to 10-character volume name that identifies the diskette. Decimal digits, uppercase and lowercase letters, and the following special characters can be used in the volume names:

```
! & , * ;
" ' ( + /
% . ) : = ?
```

FILES UTILITY SYSTEM

gran The granularity, in bytes, for this volume. The granularity is the number of bytes obtained during each diskette access. If you omit this parameter, the default volume granularity is the device granularity (the number of bytes in a physical sector). Specifying any value less than the device granularity causes the default to be used. Any non-multiple of device granularity (such as 128 or 512) is rounded upward to the next higher multiple of device granularity.

To format a double density, two sided diskette with a granularity of 512 or 1024, use the FORMAT command in the Start-up System or the iRMX 86 Human Interface subsystem.

ileave The interleave factor for the volume, or the number of physical sectors between logical sectors. You can specify any integer from 1 to 13 for this value. If you omit this parameter, a default value of 1 is assumed.

nodes The number of files that can be created on this volume. If you omit this parameter, a default value of 100 is assumed.

switch A switch that indicates the support option for this volume. One value can be entered for the switch:

NAMED The volume is created for
 the named file driver. The ROOT
 directory is initialized.

If you omit this switch, the volume is created for the physical file driver. In this case, FORMAT records the interleave information on the diskette but does not initialize any of the iRMX 86 file structures.

HELP (HE)

This command displays a list of the available Files Utility commands and their syntax on the console screen. The format is as follows:

HE

FILES UTILITY SYSTEM

UPCOPY (UC)

This command creates an iRMX 86 file and copies the specified ISIS-II file to it. If the iRMX 86 file already exists, it is written over. The format is as follows:

```
UC isis filename TO rmx pathname
```

where:

isis filename Name of the ISIS-II file to be copied.

rmx pathname Path name of the iRMX 86 file to be created.

ERROR MESSAGES

The Files Utility displays all error messages on the screen of the INTELLEC System. These messages can be in any of three forms. If the message is

UNRECOGNIZED COMMAND

the Files Utility does not recognize the spelling of your command and prompts for another command.

The Files Utility actually uses the ISIS-II operating system to read and write diskettes attached to the INTELLEC system. If the ISIS-II system detects any errors, it returns an error code to the Files Utility. Whenever the Files Utility receives an ISIS-II error, it displays the following message:

ISIS ERROR # nnn

where nnn is in decimal. To interpret this error message, refer to the ISIS-II USER'S GUIDE. Fatal errors require you to restart the Files Utility System by using the FILES.CSD file, as described earlier in this chapter.

FILES UTILITY SYSTEM

When reading or writing on drives attached to the iSBC 86/12A board, the Files Utility System uses the iRMX 86 Nucleus and the iRMX 86 I/O System. If either of these modules returns an exceptional condition code to the Files Utility, the following message is displayed:

RMX EXCEPTION # mmmm

where mmmm is in hexadecimal. For a brief explanation of such an error message, refer to Appendix B. For more detailed information, refer to the iRMX 86 NUCLEUS REFERENCE MANUAL, the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL, or the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL. After this message is displayed, the Files Utility prompts for the next command.

APPENDIX A. ORIGINAL BOARD JUMPER CONNECTIONS

This appendix contains lists of the jumper connections that are made on the iSBC 86/12A, 204, 206, 208, 215, 218, 220, and 254 boards at the factory. The iRMX 86 Operating System is designed to run with factory-jumpered boards that have been modified as described in this manual. To ensure that your boards are jumpered correctly, first restore them to the condition indicated in this appendix. Then modify them as recommended in Chapter 3 of this manual.

Table A-1. Original iSBC™ 86/12A Jumpers

<u>Pin Connections</u>			
5-6	7-8	7-10	13-14
15-16	17-18	19-20	21-25
24-35	26-27	28-29	30-31
32-33	39-40	42-43	54-55
56-57	59-60	68-76	79-83
87-89	92-93	94-96	97-98
103-104	105-106	125-126	129-130
143-144	151-152		

<u>Jumper Pad</u>	<u>Connection</u>
W1	A-B
W2	A-B
W3	A-B
W4	A-B
W5	A-B
W6	A-B
W7	A-C
W8	A-D
W9	A-C

ORIGINAL BOARD JUMPER CONNECTIONS

Table A-2. Original iSBC™ 204 Jumpers

<u>Pin Connections</u>			
1-8	19-20	23-24	26-27
37-38	45-47	55-56	63-67
75-76	77-78		
<u>Jumper Pad</u>	<u>Connection</u>		
W1	B-C		

Table A-3. Original iSBC™ 206 Jumpers (Channel Board)

4-5	9-11	13-17	15-16
-----	------	-------	-------

Table A-4 Original iSBC™ 208 Jumpers

<u>Pin Connections</u>			
E18 - E19	E21 - E22	E25 - E26	E31 - E32
E38 - E39	E45 - E49	E77 - E78	

Table A-5. Original iSBC™ 215A Jumpers

<u>Jumper Pad</u>	<u>Connection</u>	<u>Jumper Pad</u>	<u>Connection</u>
W1	1-3	W9	1-2
W3	1-2	W10	1-2
W4	1-2	W13	1-2
W5	1-3	W14	1-2
W6	1-3	W16	1-2
W7	1-3	W17	1-2
W8	1-3	W22	1-2

ORIGINAL BOARD JUMPER CONNECTIONS

Table A-6. Original iSBC™ 215B Jumpers

<u>Jumper Pad</u>	<u>Connection</u>	<u>Jumper Pad</u>	<u>Connection</u>
W1	1-2	W10	1-2
W3	1-2	W13	1-2
W4	1-2	W14	1-3
W5	1-2	W15	1-2
W6	1-2	W16	1-2
W7	1-2	W17	1-2
W8	1-2	W22	1-3

Table A-7. Original iSBC™ 218 Jumpers

<u>Jumper Pad</u>	<u>Connection</u>
W2	A-C
W8	A-B

Table A-8 Original iSBC™ 220 Jumpers

W1-2 to W1-C	W2-2 to W2-C	W3-2 to W3-C
W4-5 to W4-C	W6-1 to W6-C	E39 to E40
E41 to E42		

Table A-9. Original iSBC™ 254 Jumpers

2-3	5-6	8-9	11-12
27-28	29-30	45-46	63-64
67-68	75-76		

APPENDIX B. iRMX™ 86 CONDITION CODES SUMMARY

Table B-1 provides a list of the iRMX 86 condition codes that may be encountered during system installation processes. It is not a complete list of all possible iRMX 86 condition codes. See the appropriate iRMX 86 manual for a more detailed description of the meanings. The appropriate manual is listed in the column marked "Manual".

Table B-1. iRMX™ 86 Condition Codes

Numeric Code		Mnemonic	Manual N B E	Meaning
Hex.	Dec.			
0H	0	E\$OK	* * *	No exceptional conditions (normal)
Environmental Conditions				
1H	1	E\$TIME	* * *	A time limit (possibly a limit of zero time) expired without a task's request being satisfied.
2H	2	E\$MEM	* * *	Insufficient available memory to satisfy a task's request.
3H	3	E\$BUSY	SPM	Another task currently has access to data protected by a region
4H	4	E\$LIMIT	* * *	A task attempted an operation which, if it had been successful, would have violated a Nucleus-enforced limit.
<p>N iRMX 86 Nucleus Reference Manual B iRMX 86 Basic I/O System Reference Manual E iRMX 86 Extended I/O System Reference Manual SPM iRMX 86 System Programmer's Reference Manual</p>				

iRMX™ 86 CONDITION CODES SUMMARY

Table B-1. iRMX™ 86 Condition Codes

Numeric Code		Mnemonic	Manual N B E	Meaning
Hex.	Dec.			
Environmental Conditions (continued)				
5H	5	E\$CONTEXT	* * *	A system call was issued out of proper context.
6H	6	E\$EXIST	* * *	A token parameter has a value which is not the token of an existing object.
7H	7	E\$STATE	*	A task attempted an operation which would have caused an impossible transition of a task's state.
8H	8	E\$NOT\$CON- FIGURED	* * *	This system call is not part of the present configuration.
9H	9	E\$INTER- RUPT\$SAT- URATION	*	An interrupt task has accumulated the maximum allowable amount of SIGNAL\$INTERRUPT requests.
0AH	10	E\$INTER- RUPT\$- OVERFLOW	*	An interrupt task has accumulated more than the maximum allowable amount of SIGNAL\$INTERRUPT requests.
20H	32	E\$FEXIST	* *	File already exists.
21H	33	E\$FNEXIST	* *	File does not exist.
22H	34	E\$DEVPD	* *	Device and file driver are incompatible.
23H	35	E\$SUPPORT	* *	Combination of parameters not supported.
24H	36	E\$EMPTY\$- ENTRY	* *	The specified slot in a directory file is empty.
25H	37	E\$DIR\$END	* *	The specified slot is beyond the end of a directory file.
26H	38	E\$FACCESS	* *	File access not granted.
N		iRMX 86 Nucleus Reference Manual		
B		iRMX 86 Basic I/O System Reference Manual		
E		iRMX 86 Extended I/O System Reference Manual		

iRMX™ 86 CONDITION CODES SUMMARY

Table B-1. iRMX™ 86 Condition Codes (continued)

Numeric Code		Mnemonic	Manual N B E	Meaning
Hex.	Dec.			
Environmental Conditions (continued)				
27H	39	E\$F\$TYPE	* *	Incompatible file type.
28H	40	E\$S\$SHARE	* *	Improper file sharing requested.
29H	41	E\$S\$SPACE	* *	No space left.
2AH	42	E\$I\$IDDR	* *	Invalid device driver request.
2BH	43	E\$IO	* *	An I/O error occurred.
2CH	44	E\$FLUSHING	* *	Connection specified in call was deleted before the operation was completed.
2DH	45	E\$I\$ILLVOL	SPM	Invalidly named volume.
2EH	46	E\$DEV\$OFF-LINE	*	The device being accessed is now offline.
40H	64	E\$PREFIX\$-SYNTAX	*	The specified path starts with a colon (:) but does not contain a second, matching colon.
41H	65	E\$CANNOT\$-CLOSE	*	The Extended I/O System was not able to transfer remaining data in buffers to output device.
42H	66	E\$I\$IOMEM	*	The Basic I/O System has insufficient memory to process a request.
44H	68	E\$MEDIA	*	The device containing a specified file is not online.
45H	69	E\$LOG\$NAME-NEXIST	*	The Extended I/O System was unable to find a specified logical name in the object directories that it checks.
<p>N iRMX 86 Nucleus Reference Manual B iRMX 86 Basic I/O System Reference Manual E iRMX 86 Extended I/O System Reference Manual SPM iRMX 86 System Programmer's Reference Manual</p>				

iRMX™ 86 CONDITION CODES SUMMARY

Table B-1. iRMX™ 86 Condition Codes (continued)

Numeric Code		Mnemonic	Manual N B E	Meaning
Hex.	Dec.			
Programmer Errors				
8000H	32768	E\$ZERO\$- DIVIDE	*	A task attempted to divide by zero.
8001H	32769	E\$OVER-FLOW	*	An overflow interrupt occurred.
8002H	32770	E\$TYPE	* * *	A token parameter referred to an existing object that is not of the required type.
8003H	32771	E\$BOUNDS	*	A task attempted to access beyond the end of a segment.
8004H	32772	E\$PARAM	* * *	A parameter which is neither a token nor an offset has an invalid value.
8005H	32773	E\$BAD\$CALL	* *	The I/O System code has been damaged, probably due to a bug in an application task. Recovery is not possible.
8020H	32800	E\$IFDR	* *	Invalid file driver request.
8021H	32801	E\$NOUSER	* *	No default user.
8022H	32802	E\$NO\$PREFIX	* *	No default prefix.
8040H	32832	E\$NOT\$PREFIX	*	Specified object not a device connection or file connection.
8041H	32833	E\$NOT\$DEVICE	*	A token parameter referred to an existing object that is not, but should be, a device connection.
8042H	32834	E\$NOT\$CON- NECTION	*	A token parameter referred to an existing object that is not, but should be, a file connection.
N		iRMX 86 Nucleus Reference Manual		
B		iRMX 86 Basic I/O System Reference Manual		
E		iRMX 86 Extended I/O System Reference Manual		

APPENDIX C. HARDWARE REQUIREMENTS FOR CUSTOM CONFIGURATIONS

The minimum hardware requirements for installing the iRMX 86 Nucleus on custom boards built from components are defined below:

- iAPX 86.
- 8253 Programmable Interval Timer (PIT).
- 8259A Programmable Interrupt Controller (PIC).
- 24K bytes ROM (for a fully-configured Nucleus).
- 2600 bytes RAM, of which 1024 bytes must be contiguous and must start at address 0.

To run the Terminal Handler or Debugger, an 8251A Programmable Communications Interface (PCI) is required, as well.

APPENDIX D. iRMX™ 86 SOFTWARE VERSION NUMBERS

The version numbers for all required and optional software that comprise Release 3.0 of the iRMX 86 Operating System are listed in Table D-1.

Table D-1. iRMX™ 86 Software Version Numbers

iRMX 86 Module	Version Number
Nucleus	4.0
Terminal Handler	3.0
Debugger	3.0
Basic I/O System	3.1
Extended I/O System	1.0
Application Loader	2.1
Bootstrap Loader	3.0
Human Interface	1.1
Files Utility	3.0
Patching Utility	1.0 & 2.0
Start-Up System	2.0
Disk Verify	1.0
Disk Backup	1.0
Disk Restore	1.0
Universal Development System Interface	1.1

INDEX

Primary references are underscored.

8255 parallel I/O port 3-3

ABS function 4-11
altering TBASIC statement lines 4-4
Application Loader 1-1, 2-2, 2-4, 7-2
application-dependent requirements 2-3
application system 2-1, 4-1, 5-1, 7-1
arrays 4-4
 storage space 4-12
ASM86 Macro Assembler 2-2, 6-1
ASM86 Software Development Utilities 2-2
ATTACHDEV command 3-8, 8-3
ATTACHDEVICE command 3-8, 5-4

backplane 3-6
base address
 iSBC 204 3-4
 iSBC 206 3-4
 iSBC 208 3-5
 iSBC 254 3-6

BASIC
 functions 4-11
 statements 4-7
Basic I/O System 1-1, 2-4, 3-3
baud rate 4-1
board
 arrangement in the chassis 3-6
 jumping A-1
 modifications 3-1
Bootstrap Loader 1-1, 1-2, 2-3, 2-4, 7-2
BPRN/ signal 3-7
BREAK command 8-4
BS1.CSD file 5-3
burn into PROM 2-3, 7-2
bus master priority 3-7

cable connections 3-7
CATALOG statement 4-13
changing diskettes 8-3
chassis arrangement 3-6
COMMON attribute 6-2
components iii, 2-2, 3-1, C-1
condition codes 8-9, B-1
constants 4-5
COPY command 5-4
CREATEDIR command 5-4, 8-4

INDEX (continued)

CRTMBOX function 4-13
CRTSEGM function 4-14
CRTSEMA function 4-14
CRTTASK function 4-15
custom boards iii, C-1

DATE command 5-5
Debugger 1-1, 1-2, 2-4, 3-7, 4-1, 7-1
deferred mode 4-3
DEBGGR.DMO 4-2
DEBUG command 5-5
DELETE command 5-4, 8-4
DELMBOX statement 4-16
DELSEGM statement 4-16
DELSEMA statement 4-17
DELTASK statement 4-17
demonstration system 4-1
DETACH command 8-5
DETACHDEVICE command 5-4
development
 environment 2-1
 process 2-2, 7-1
Development System 1-1, 2-2, 3-8, 4-1, 5-2, 6-2, 8-1
dictionary of statements and functions 4-6
DIR command 5-4, 8-5
disk drives 2-2, 3-9, 3-10
disk identifier 8-6
diskettes 1-3, 2-2, 5-1, 8-1, 8-3, 8-6
DOWNCOPY command 5-4, 8-5

error messages 6-6, 8-8
expressions 4-5
Extended I/O System 1-1, 2-4

factory-installed jumpers 3-1, A-1
file management commands 5-4, 8-3
Files Utility System 1-1, 1-2, 3-8, 5-1, 7-1, 8-1
FILES.CSD file 8-2
FOR statement 4-7
FORMAT command 5-4, 8-6
functions 4-11

general requirements 2-2
GETTKNS function 4-17
GOSUB statement 4-8
GOTO statement 4-8
granularity 8-6

hardware
 considerations 2-3, 3-1
 requirements 4-1, 5-2, 7-1, 8-1, C-1
header records 6-1, 6-2, 6-6
HELP command 8-7
Human Interface 1-1, 2-4, 5-1, 6-2, 7-2, 8-7

INDEX (continued)

iAPX 86, 88 1-1, 5-3
 ICE-86 In-Circuit Emulator 2-2, 2-3, 3-7, 3-8, 4-2, 7-1
 iCS 80 chassis 3-6, 3-7
 IF statement 4-8
 INTELLEC Development System 1-1, 2-2, 2-3, 3-8, 4-1, 5-2, 6-2, 8-2
 in-place patch 6-1, 6-3, 6-4
 INPUT statement 4-9
 interactive mode 4-3, 4-10
 interleave factor 8-7
 interrupt levels 3-2, 3-3, 3-4, 3-5, 3-6
 introduction to the RMX/86 package 1-1
 INVALID MODULE TYPE error message 6-6
 INVALID RECORD TYPE error message 6-7
 INVALID SYNTAX error message 6-7
 inventory 1-2
 iRMX 86
 development environment 2-1
 operating system 1-1
 package 1-1
 software version numbers D-1
 statements and functions 4-12
 iSBC 032/048/064 RAM board 3-6
 iSBC 016A/032A/064A/028A/056A RAM board 3-6
 iSBC 204
 board 2-3, 3-1, 3-9, 5-2, A-2
 modifications 3-3
 iSBC 206
 board 2-3, 3-1, 3-7
 modifications 3-4
 iSBC 208
 board 2-3, 3-1, 3-9, 5-2, A-2
 modifications 3-5
 iSBC 215
 board 2-3, 3-1, 3-10, 5-2, A-2
 modifications 3-5
 iSBC 220
 board 2-3, 3-1, 3-10, A-3
 modifications 3-6
 iSBC 254
 board 2-3, 3-1, 3-10, A-3
 modifications 3-6
 iSBC 604 cardcage/backplane 3-6
 iSBC 614 expansion cardcage/backplane 3-6
 iSBC 655 chassis 3-6
 iSBC 660 chassis 3-6
 iSBC 86/12A
 board 2-3, 3-1, 3-7, 3-8, 5-2, 8-1, A-1
 modifications 3-2
 iSBC 902 terminator pack 3-3
 iSBC 957A/B package 2-3, 3-2, 4-2, 5-2, 7-1, 7-2, 8-1, 8-2
 iSBX 218
 board 2-3, 3-1, 3-9, 5-2, A-3
 modifications 3-5
 iSBC 337 multimodule 3-2
 ISIS-II errors 8-8
 ISIS-II operating system 2-2, 8-1, 8-8

INDEX (continued)

J1 pin assignments 3-3
jump instruction patch 6-3
jumper connections 3-1, 3-6, A-1

LET statement 4-9
library module patching 6-3
line printer 3-3
LINK86 command 6-2, 6-3, 6-5
LIB86 command 6-5
LIST statement 4-10
loading the demonstration system 4-2
LOOKUPO function 4-18

mailbox 4-13, 4-16, 4-19, 4-21
manuals 1-2, 3-1
memory board jumper connections 3-6
memory requirements 2-4, 5-2, 7-1
messages 4-21, 8-8
modes 4-3
MULTIBUS contention 3-5

named file driver 8-7
NEW statement 4-10
NEXT statement 4-7
Nucleus 1-1, 1-2, 2-4, 4-1, C-1
Nucleus Demonstration System 4-1
NUCLUS.DMO 4-2

object 4-13, 4-17, 4-19, 4-21
object directory 2-4, 4-13, 4-18, 4-23
operating modes 4-3

Patching Utility 1-1, 6-1
PATCH.CSD file 6-4
physical device names 3-4, 3-8, 3-9, 3-10
PL/M-86 compiler 2-2, 6-2
priority 3-6, 3-7
PRINT statement 4-10
PROG/ logical name 5-5
program storage 4-3
protecting system software 1-3
PTCH86 command 6-2, 6-5
PTCH86.R86 file 6-2, 6-3
PUBLIC attribute 6-2, 6-3, 6-5

RAM requirements 2-3, 2-4, C-1
RCVUNIT function 4-19
recommendations 1-3
RECVMSG function 4-19
REM statement 4-11
RENAME command 5-4
replacement code 6-1
RESTASK statement 4-20
RETURN statement 4-11

INDEX (continued)

RND function 4-12
ROM requirements 2-3, 2-4, C-1
ROOTJB.DMO 4-2
RS232 interface 2-3, 4-1
RUN statement 4-11

segment 4-14, 4-16
SEGMENT NOT FOUND error message 6-7
semaphore 4-14, 4-15, 4-17, 4-19, 4-22
SENDMSG statement 4-21
serial priority scheme 3-7
SGEN.CS file 5-5
SIZE function 4-12
SLEEP statement 4-21
SMAKE.CSD file 5-5
SNDUNIT 4-22
starting the Files Utility 8-2
Start-Up System 1-1, 1-2, 1-3, 2-3, 3-1, 3-8, 5-1, 7-1, 8-7
STARTUPCOPY.CSD file 5-5
statements and functions 4-5, 4-12
 dictionary 4-6
STOP statement 4-11
storing programs 4-3
SUBMIT files
 BS1 5-3
 FILES 8-2
 PATCH 6-5
 SUPLD 5-3
 SGEN 5-5
 SMAKE 5-5
 STARTUPCOPY 5-5
SUPLD.CSD file 5-3
support option 8-7
SUSTASK statement 4-22
SYSTEM/ logical name 5-5
system software protection 1-3

target system 2-2, 4-1
task 4-15, 4-17, 4-20, 4-21, 4-22
TBASIC.DMO 4-2
TBASIC interpreter 4-1, 4-3, 4-5
Terminal Handler 1-1, 1-2, 2-4, 3-2, 3-7
TIME command 5-5
token 4-13, 4-14, 4-15, 4-17, 4-18, 4-19
translator header record 6-1, 6-2, 6-6

UNCATLG statement 4-23
units 4-19, 4-22
UNRECOGNIZED command 8-8
UPCOPY command 3-8, 5-4, 8-7
using the demonstration system 4-3
using the Files Utility 8-3

INDEX (continued)

variables 4-4
version numbers D-1
volume
granularity 8-6
name 8-6

wake-up address
iSBC 215 3-5
iSBC 220 3-6



REQUEST FOR READER'S COMMENTS

Intel Corporation attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply.

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.



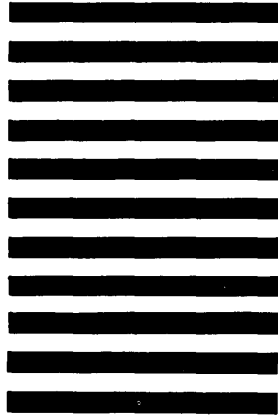
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Pkwy.
Hillsboro, Oregon 97123

O.M.S. Technical Publications





INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 987-8080

Printed in U.S.A.