RMX/86™ INSTALLATION GUIDE FOR ISIS-II USERS

Manual Order Number 9803125-01

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

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.

The following are trademarks of Intel Corporation and may be used only to describe Intel products:

BXP	Intellec	Multibus
i	iSBC	Multimodule
ICE	iSBX	PROMPT
iCS	Library Manager	Promware
Insite	MCS	RMX
Intel	Megachassis	UPI
Intelevision	Micromap	μScope

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

PREFACE

The RMX/86 Operating System is a software package that provides a real-time, multitasking environment for Intel iSBC 86/12A single board computers. This manual provides the information that you, as a new RMX/86 user, need in order to put together your first system. It does the following:

- Introduces the Operating System and shows how it is packaged.
- Defines the hardware and software environment in which application systems are developed.
- Describes the modifications to hardware needed to support the RMX/86 Operating System and shows how to put the various hardware pieces together.
- Explains how to install and use the demonstration system. This is a tested and debugged RMX/86 application system.
- Describes the Files Utility System. This system is used to format RMX/86 disks, create and delete files, and transfer information between ISIS-II disks and RMX/86 disks.
- Outlines the process of developing an RMX/86-based application system.

READER LEVEL

This manual assumes that you are familiar with the following:

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

It also assumes that you have read the Introduction to the RMX/86 Operating System.

RELATED PUBLICATIONS

The following manuals provide additional background and reference information.

Manual	Number
Introduction to the RMX/86 Operating System	9803124
RMX/86 Configuration Guide for ISIS-II Users	9803126
RMX/86 Nucleus, Terminal Handler, and Debugger Reference Manual	9803122
RMX/86 I/O System Reference Manual	9803123
RMX/86 Configuration Guide for ISIS-II Users	9803126
ISIS-II User's Guide	9800306
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
iSBC 86/12A Single Board Computer Hardware Reference Manual	9803074
iSC 80 Industrial Chassis Hardware Reference Manual	9800799
iSBC 660 System Chassis Hardware Reference Manual	9800505
iSBC 204 Flexible Diskette Controller Hardware Reference Manuoal	9800568
iSBC 206 Disk Controller Hardware Reference Manual	9800567
iSBC 032/048/064 Random Access Memory Boards Hardware Reference Manual	9800488

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF FIGURES

TABLE OF TABLES

CHAPTER 1. INTRODUCTION TO THE RMX/86 PACKAGE

The RMX/86 Operating System is a real-time, multitasking operating system for iSBC 86/12A single board computers. It consists of a Nucleus and several optional parts: the Terminal Handler, the Debugger, and the I/O System. The Nucleus is the core of the Operating System, coordinating the activities of the rest of the system. The Terminal Handler provides a terminal interface to the application system. The Dubugger provides monitoring and debugging capabilities; and the I/O System adds file and device access capabilities. The software that you write runs under the supervision of the Nucleus and in con- junction with one or more of the optional parts.

Physically, the RMX/86 Nucleus and Operating System parts reside on diskettes in your RMX/86 package and are in the form of libraries of relocatable code. In order to create an RMX/86 application system, you must use an INTELLEC Development System to combine this RMX/86 code with your application code and produce object code that executes on an iSBC 86/12A single board computer.

INVENTORY

Your shipment of RMX/86 materials includes a number of items, among which are six manuals and ten diskettes. The manuals are:

- RMX/86 Installation Guide for ISIS-II Users - This manual which you are now reading helps you to make specific RMX/86-required hardware modifications, install and run the demonstration system, and use the RMX/86 file utilities.

- Introduction to the RMX/86 Operating System - This manual introduces you to the RMX/86 product. Read this manual before any other.

- RMX/86 Nucleus, Terminal Handler, and Debugger Reference Manual - This manual is the primary reference source for the Nucleus, Terminal Handler, and Debugger.

- RMX/86 I/O System Reference Manual - This manual is the primary reference source for the I/O System.

- RMX/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.

- RMX/86 Configuration Guide for ISIS-II Users - This manual describes how to build a software application system by combining the Operating System and application software.

Five of the ten diskettes in the RMX/86 package are double-density diskettes. Each contains libraries, submit files, and other associated files for the Operating System, the utilities, or the demonstration system. They are:

- Nucleus diskette
- Terminal Handler and Debugger diskette
- I/O System diskette
- Utilities diskette
- Demonstration system diskette

The other five diskettes are single-density versions of the first five.

RECOMMENDATIONS

Because of the ever-present possibility of accidents, make at least one backup copy of each diskette that you are planning to use. Keep the Intel-supplied diskettes as masters. Use the copies for system development.

CHAPTER 2. THE RMX/86 DEVELOPMENT ENVIRONMENT

The development of an RMX/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 depicts a typical development hardware environment.

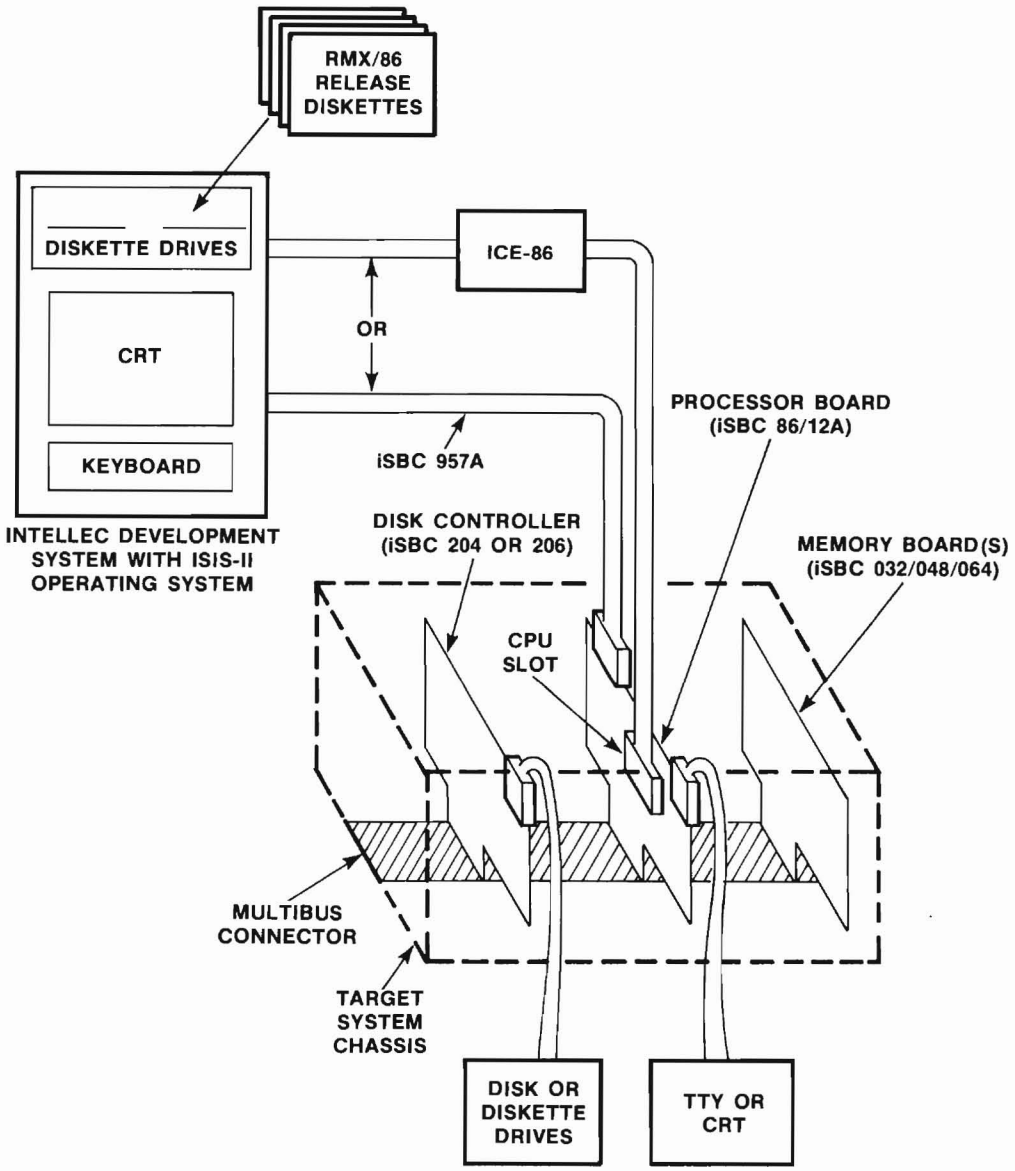

Figure 2-1. RMX/86 Example Development Environment

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. It also shows some typical devices attached to the target system. Although it does not depict all of the hardware devices that can be attached to a target system, it does show some of the more common ones.

GENERAL REQUIREMENTS

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

Development System

- An INTELLEC Development System with CRT, keyboard, at least two disk drives, and at least 64K of RAM.
- A diskette containing the ISIS-II Operating System.
- An MCS-86 Macro Assembler and/or PLM/-86 compiler, as well as the MCS-86 Software Development Utilities, on diskette.
- Diskettes containing the required Intel-supplied RMX/86 software.

Target System

- An iSBC 86/12A processor board, which is the heart of the application system.
- A chassis to hold and supply power to the processor board and any other boards of the system.
- Enough memory to contain the Nucleus and any other Operating System parts and application jobs.

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

- The ICE-86 In-Circuit Emulator
- The iSBC 957A Interface and Execution Package

Both of these products transfer code from diskette on the Development System to RAM on the target system. After you have tested the code, you can burn it into PROM, and place the PROM on the target system, eliminating the need for the ICE-86 emulator or the iSBC 957A package to load the code.

APPLICATION-DEPENDENT REQUIREMENTS

You may have to add additional hardware to your target system, depending on the type of application you have.

- If your application includes the Terminal Handler or the Debugger, connect an RS232 protocol terminal to the serial I/O port of the processor board.
- If your application includes the I/O System, place at least one controller board in the chassis with the processor board. The iSBC 204 and 206 boards can be used. Connect the controllers to their associated disk drives.

If your application system includes the I/O System, you must also use the Files Utility System to format disks for use in your target system disk drives. The Files Utility System requires the use of the iSBC 957A package.

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

Table 2-1. Memory Requirements

	ROM (in bytes)	RAM (in bytes)
Nucleus	28K	300H
Terminal Handler	3.3K	
Debugger	24K	
I/O System	50K	16K
Files Utility System		192K

The figures shown in Table 2-1 are estimates. The I/O System RAM requirement, for example, depends on the number of connections desired. A larger amount of available RAM permits more connections.

CHAPTER 3. HARDWARE CONSIDERATIONS

This chapter discusses the hardware modifications and installation procedures that apply specifically to RMX/86 systems. It discusses:

- 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 specific instructions on how to modify hardware components, it contains only that information that applies directly to the RMX/86 Operating System. You will have to make other hardware modifications in addition to those described here. For the other modifications refer to the appropriate hardware manuals for the components of your system.

BOARD MODIFICATIONS

Before you insert your processor board, controller boards, and memory boards into the chassis, you must modify certain jumper connections and switch settings on these boards. The following sections discuss these modifications.

iSBC 86/12A JUMPER CONNECTIONS

Each iSBC 86/12A board comes from the factory with certain jumper posts already connected. Appendix A contains a list of the jumper connections that were made at the factory. Compare your board with Appendix A to verify that all default connections are present. The iSBC 86/12A Single Board Computer Reference Manual describes these connections and the options available in detail .

The following sections describe the modifications that you must make to the iSBC 86/12A board in order to load and run your RMX/86 software.

Jumper Connections for the iSBC 957A Package

If you are going to use the iSBC 957A package in conjunction with the Files Utility System or to load your software into

RAM, refer to the iSBC 957A INTELLEC--iSBC 86/12A Interface and Execution Package User's Guide. It describes the jumper connection modifications you must make to the iSBC 86/12A board in order to use the iSBC 957A package.

Jumper Connections for the Terminal Handler and the Debugger

If you are including the Terminal Handler or the Debugger in your application system, you must connect interrupts 6 and 7 to the 8251A USART. To do this, connect the following jumper posts on the iSBC 86/12A board:

E82 - E75
E90 - E74

Jumper Connections for the I/O System

In order to use the I/O System, you must ensure that the interrupts sent by the controller boards are received by the 8251A USART and correspond to the interrupt levels of the controller board. The released version of the I/O System configuration file assumes that the iSBC 204 controller uses interrupt level 5 and the iSBC 206 controller uses interrupt level 4. If you use these controllers in your system, make the following jumper connections on the iSBC 86/12A board to connect the interrupt signals.

Controller	iSBC 86/12A Jumper Connections	Interrupt Level
iSBC 204 board	E68-E76 (default)	5
iSBC 206 board	E69-E77	4

If you use both controllers in your system, make both jumper connections. The iSBC 86/12A single Board Computer Hardware Reference Manual describes this process in detail. To set the interrupt levels on the controller boards, refer to the next two sections of this manual.

iSBC 204 JUMPER CONNECTIONS

When using the iSBC 204 controller to interface to the I/O System, make sure that the interrupt request signal of the controller is assigned to the interrupt level specified for it in the device information table of the I/O System configuration file. To specify the interrupt level in the device information table, refer to the RMX/86 Configuration Guide for ISIS-II Users. The values in the device information table correspond to interrupt levels as follows:

Interrupt level	Device Information Table Entry
0	08H
1	018H
3	038H
4	048H
5	058H

To set the interrupt request signal of the controller, remove the connector from the default jumper connection (63-67) and make one of the following connections, depending on the interrupt level desired.

Interrupt Level	Jumper Connection
0	61-65
1	62-66
3	64-68
4	72-68
5	71-67

Do not use interrupt levels 2, 6, or 7. The clock uses level 2 and the Terminal Handler uses levels 6 and 7.

Also make the following jumper connections on the iSBC 204 board:

75-76
77-78

You must also select an I/O base address by setting switch S2 on the iSBC 204 board. The released I/O System configuration file assumes an iSBC 204 base address of OAOH. Select this value by setting switch S2 as follows:

OAOH	S2 Switch Position Callout			
	7	6	5	4
Position	ON	OFF	ON	OFF

Refer to the iSBC 204 Flexible Diskette Controller Hardware Reference Manual for more information on the iSBC 204 controller.

iSBC 206 CONTROLLER SWITCH SELECTION

When using the iSBC 206 controller to interface to the I/O System, make sure that the controller interrupt signal is assigned to the interrupt level specified for it in the device information table of the I/O System configuration file. To specify the interrupt level in the device information table, refer to the RMX/86 Configuration Guide for ISIS-II Users. The values in the device information table correspond to interrupt

as shown in the "iSBC 204 Jumper Connections" section of this chapter. To set the controller interrupt signal, place the iSBC 206 channel board rotary switch (S2) in the position corresponding to the interrupt level. The switch positions (0 through 7) are silk-screened on the board and select the corresponding interrupt levels. Do not use interrupt levels 2, 6, or 7. The clock uses level 2, and the Terminal Handler uses levels 6 and 7. The released I/O System configuration file assumes that the iSBC 206 controller uses interrupt level 4.

You must also select an I/O base address by setting switch S1 on the iSBC 206 channel board. The I/O System configuration file assumes an iSBC 206 base address of 068H. Select this value by setting switch S1 as follows:

68H	S1 Switch Position Callout					
	5	6	7	8	3	4
Position	OFF	ON	ON	OFF	ON	OFF

Refer to the iSBC 206 Disk Controller Hardware Reference Manual for further information about the iSBC 206 controller.

MEMORY BOARD JUMPER CONNECTIONS

Connect the jumpers on your iSBC 032/048/064 RAM boards to assign memory to the correct contiguous memory locations. Refer to the iSBC 032/048/064 Random Access Memory Boards Hardware Reference Manual for the procedure to do this.

Both the memory boards that you insert in the chassis and the dual port RAM on the iSBC 86/12A board can be shared with other bus masters (such as controller boards) via the MULTIBUS interface. The iSBC 86/12A board always assumes that its dual port RAM begins at address zero. You can, however, specify how this dual port RAM appears to other bus masters by connecting jumpers (E113 through E128) and setting switch S1 on the iSBC 86/12A board. The iSBC 86/12A Single Board Computer Reference Manual contains instructions for assigning dual port RAM addresses. When you assign these addresses, make sure that the dual port RAM addresses do not overlap other memory board addresses, or you could damage your system. It is recommended that you assign dual port RAM to the same addresses on the MULTIBUS as are assumed by the iSBC 86/12A board (that is, starting at address zero).

BOARD ARRANGEMENT IN THE CHASSIS

When building your hardware system, use the serial priority scheme for resolving MULTIBUS contention. This necessitates grounding the BPRN/ signal of the highest priority bus master. Bus masters are boards that are capable of acquiring and controlling the MULTIBUS interface. Even if you have only a single board in your system (the iSBC 86/12A board) you must ground its BPRN/ signal.

If you place your boards in the iSBC 604 and iSBC 614 cardcage/backplanes of the iSBC 660 or iCS 80 chassis, you can implement the serial priority resolution scheme by connecting jumpers on the backplane. If you ground BPRN/ of the top slot (J2) of the cardcage, the priority of the slots is sequential, with J2 having the highest priority and J5 having the lowest. This is described in the iSBC 86/12A Hardware Reference Manual as well as in the manuals for the chassis and the controller boards. You must alter this slot priority, depending on how you arrange the boards in the backplane.

The order in which you arrange the boards in the backplane depends on the type and number of bus masters in your system, and their physical placement requirements. The iSBC 86/12A board, the iSBC 204 board, and the iSBC 206 channel board are all bus masters. Memory boards are not bus masters; you can place them in any available slot. The serial priority scheme supports a maximum of three bus masters.

Physical requirements include placing the iSBC 206 channel board and interface board in adjacent slots, placing the iSBC 86/12A board in the top slot if using the ICE-86 In-Circuit Emulator, and placing the iSBC 86/12A board in the slot that physically accomodates it. (If multimodules are attached to the iSBC 86/12A board it may not physically fit in a slot directly below another board.)

The following sections describe arranging the boards in the cardcage and installing the proper jumper connections for various combinations of bus masters.

NOTE

Before you install jumper connections on the chassis or insert the boards, make sure that the power to the chassis is off.

iSBC 86/12A BOARD ALONE

Place the iSBC 86/12A board in the top slot (J2) of the cardcage. Place memory boards in any of the other slots. Connect the jumper posts on the etch side of the backplane as follows:

B-L

This designates J2 as the highest priority slot; thus the iSBC 86/12A board is the highest priority bus master.

iSBC 86/12A BOARD WITH iSBC 204 BOARD

Place the iSBC 86/12A board in the top slot (J2) of the cardcage. Place the iSBC 204 board in J3. Place memory boards in any of the other slots. Connect the jumper posts on the etch side of the backplane as follows:

C-L
E-B

This designates J3 as the highest priority slot and J2 as the next highest. Thus the iSBC 204 board is the highest priority bus master and the iSBC 86/12A board is the second highest.

iSBC 86/12A BOARD WITH iSBC 206 BOARDS

Place the iSBC 86/12A board in the top slot (J2) of the cardcage. Place the iSBC 206 channel board in J3 and the iSBC 206 interface board in J4. Place memory boards in any of the other slots. Connect the jumper posts on the etch side of the backplane as follows:

C-L
E-B

This designates J3 as the highest priority slot and J2 as the next highest. Thus the iSBC channel board is the highest priority bus master and the iSBC 86/12A board is the second highest.

iSBC 86/12A BOARD WITH BOTH iSBC 204 and iSBC 206 BOARDS

Place the iSBC 204 board in the top slot (J2) of the top (614) cardcage. Place the iSBC 206 channel board in J3, the iSBC 206 interface board in J4, and the iSBC 86/12A board in the top slot (J2) of the bottom (604) cardcage. Place memory boards in any of the other slots. Connect the jumper posts on the etch side of the top backplane as follows:

B-L
E-K

This designates J2 as the highest priority slot, J3 as the second highest, and the top slot of the bottom cardcage (J2) as the third highest. Thus the iSBC 204 board is the highest priority bus master, the iSBC 206 channel board is the second highest, and the iSBC 86/12A board is the third highest.

CABLE CONNECTIONS

After you have placed the boards in the chassis, install the cables joining the boards to the various parts of the system. If you are using the Terminal Handler or Debugger, connect the terminal of your RMX/86 system to the serial I/O port of the iSBC 86/12A board. The iSBC 86/12A Single Board Computer Hardware Reference Manual discusses this process. If you are using the I/O System and have an iSBC 204 controller, connect the drives to their associated I/O connectors (J1 or J2) on the iSBC 204 board. The iSBC 204 Flexible Diskette Controller Hardware Reference Manual discusses this process. If you are using the I/O System and have an iSBC 206 controller, connect the iSBC 206 interface board to the drives as described in the iSBC 206 Disk Controller Hardware Reference Manual.

Also connect the cables for the ICE-86 In-Circuit Emulator or the iSBC 957A 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 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. Refer to the iSBC 957A INTELLEC -- iSBC 86/12A Interface and Execution Package User's Guide for a description of this process.

CHAPTER 4. THE DEMONSTRATION SYSTEM

Two of the disks shipped with the RMX/86 package contain the demonstration system. One is a single-density disk and one is a double-density disk, but both contain the same information. Therefore, use the one that is appropriate for the disk drives on your Development System.

The demonstration disk 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 RMX/86 objects. This chapter describes what you need in order to run the demonstration system, how to load the system, and how to use it.

HARDWARE REQUIREMENTS

In order to run the demonstration system, your system must contain the following equipment:

- An INTELLEC Microcomputer Development System with CRT, keyboard, and at least 2 disk drives
- An iSBC 86/12A board and chassis
- 96K of contiguous RAM, starting at address 0, for use with the iSBC 86/12A board
- A RS232 protocol terminal
- Either:

  An ICE-86 In-Circuit Emulator
  or
  An iSBC 957A package

You need the terminal in order to communicate with the application system, and either the ICE-86 emulator or the iSBC 957A 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 for Debugger use. Refer to Chapter 3 for the jumpering information.

The demonstration system assumes that your terminal operates at 9600 baud. If it operates at a different baud rate, you must reconfigure the Debugger, specifying the correct rate. Refer to the RMX/86 Configuration Guide for ISIS-II Users for procedures to do this.

LOADING THE DEMONSTRATION SYSTEM

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

```
NUCLUS.DMO
DEBUGR.DMO
TBASIC.DMO
ROOTJB.DMO
```

Load the file containing the root job, ROOTJB.DMO, last because it contains instructions which initialize 8086 registers to their proper values.

In order to load and start the demonstration system with the ICE-86 emulator, place a system disk containing ICE-86 software in drive F0 of the Development System and the demonstration system diskette in drive F1. Enter the following commands at the keyboard of the Development System:

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

Refer to the ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users for complete instructions on the use of the ICE-86 emulator.

In order to load and start the demonstration system with the iSBC 957 package, place a system disk containing the iSBC 957A software in drive F0 of the Development System and the demonstration system diskette in drive F1. Enter the following commands at the keyboard of the Development System:

```
SBC86l
L    :F1:NUCLUS.DMO
L    :F1:DEBUGR.DMO
L    :F1:TBASIC.DMO
L    :F1:ROOTJB.DMO
G
```

Refer to the iSBC 957A INTELLEC -- iSBC 86/12A Interface and Execution Package User's Guide for complete instructions on the use of the iSBC 957A package.

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

USING THE DEMONSTRATION SYSTEM

After you have initiated execution of the demonstration system, the TBASIC interpreter displays a message 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 whole. The TBASIC interpreter also contains commands and functions which perform such RMX/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, direct and interpretive. In direct mode you enter individuial statements; the interpreter processes them and displays the results at the terminal as they are entered. Certain statements, such as LIST and RUN, are valid only in direct mode. Direct mode is the initial mode of the interpreter.

In interpretive mode, you run entire programs. You enter and change programs in direct mode and then transfer to interpretive mode to run them. You transfer to interpretive mode by entering the RUN statement.

STORING PROGRAMS

You can create and store an entire program by prefacing each statement you enter with a line number. This prevents the interpreter from immediately processing the statement. All statements prefaced with line numbers are stored in memory. When you enter the RUN statement, the entire program is interpreted in line number order (not necessarily the order in which you entered it).

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.

If you do not preface a statement with a line number, the interpreter processes it immediately.

VARIABLES

TBASIC supports two kinds of variables. A variable can either be a single alphabetic character (A-Z) or an array element. The intepreter 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 this 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
                  described later in this chapter
```

The following are not acceptable as variable names.

```
5
AB
2C
INTEG
F(2)
```

CONSTANTS

The interpreter supports integer constants in the range -32768 to +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 some perform RMX/86 Nucleus operations. The following sections describe these statements and functions. A statement and function dictionary appears first, listing 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.

STATEMENT AND FUNCTION DICTIONARY

BASIC STATEMENTS

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

LIST

This statement lists part or all of the program currently in memory. You can enter this statement in direct mode only. Its 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, LIST lists the entire program.

NEW

This statement clears memory of all source statements. You can enter this statement in direct mode only. Its format is as follows:

```
NEW
```

RUN

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

```
RUN
```

PRINT

This statement prints a line at the console. Its 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 six 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 with commas, can be printed on the same line.

If you enter the PRINT statement without any parameters, the interpreter prints a blank line.

FOR ... NEXT

These statements provide looping control. Their 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.
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 interpreter assumes a default value of 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. Its format is as follows:

```
GOSUB line-number
```

where:

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

Subroutines may be recursive.

RETURN

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

```
RETURN
```

Subroutines may be recursive.

LET

This statement assigns a value to a variable. Its 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.

IF

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

```
IF condition statement
```

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 which is executed only if condition is true.

GOTO

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

```
GOTO line-number
```

where:

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

REM

This statement allows you to place remarks in your source code. Its format is as follows:

REM comment

where:

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

INPUT

This statement allows you to assign a value to a variable from the console while the program is running. Its format is as follows:

INPUT var-name

where:

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

During execution, the interpreter prompts the terminal with the following message:

var-name ?

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

STOP

This statement stops the execution of the program. Its 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. These functions can be used anywhere TBASIC expressions can be used.

RND

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

RND (expression)

where:

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

ABS

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

ABS (expression)

where:

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

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, if you have a large program you are allowed fewer array elements than if you have a small program. The format of this function is as follows:

SIZE

RMX/86 STATEMENTS AND FUNCTIONS

This section describes the TBASIC statements and functions that allow you to make RMX/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 Nucleus, Terminal Handler, and Debugger Reference Manual for a complete description of the parameters.

For many of these statements and functions, you must supply a variable in which the interpreter returns the status of the RMX/86 system call. The interpreter has a convention that any variable in which it returns a value must be preceded by a period (.) in the statement declaration. You need not include the period when you originally define or later reference the variable.

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

CRTTASK

This function creates a task and returns a token for that task. The format of this function 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 starting address. The atsign (@) must precede this value.
data$seg	Base value of the task data segment. A zero indicates that the task initializes the data segment register.
stack$ptr	Pointer indicating the address of the stack segment. A value of 0:0 indicates that the Nucleus assigns a stack segment. An atsign (@) must precede this parameter.
stack$size	Size of the stack.
stat$var	Variable in which the interpreter returns the status of the create operation.

Example:

The following function call creates a task with a priority of 129, start address of 14A0:343 (obtained from the locate map for that task), and 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)
```

DELTASK

This statement deletes a task. Its 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 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)
```

SUSTASK

This statement suspends a task. Its 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)
```

RESTASK

This statement resumes a suspended task. Its 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)
```

SLEEP

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

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

where:

units	Number of 10-millisecond units that the calling task is willing to sleep. A value of zero places the calling 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)
```

GETTKNS

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

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

where:

obj$token	Token for the requested object.

select$val	Value indicating the object for which a token is requested. Possible values include: 0 Token for the interpreter task. 1 Token for the interpreter task's job. 2 Token for the interpreter job's parameter object. 3 Token for the root job.
stat$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)
```

CATALOG

This statement catalogs a given object in a given directory. Its 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 value 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)
```

LOOKUPO

This function looks up a name in an object directory and returns a token for that object. The format of this function 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 value of 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$limit	Number of 10-millisecond time units that the function is willing to wait for the name to become available.
stat$var	Variable in which the interpreter returns the status of the look up operation.

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 returns a token for the object in variable T. It returns the status of the operation in variable S.

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

UNCATLG

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

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

where:

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

Example:

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

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

CRTSEMA

This function creates a semaphore. The format of the function 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.
max$value	Maximum value of the semaphore.
sema$flags	Value indicating the type of semaphore. Possible values include: 0 First-in, first-out semaphore 1 Priority semaphore

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

Example:

The following function call creates a priority semaphore 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)
```

DELSEMA

This statement deletes a semaphore. The format of this statement 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 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)
```

SNDUNIT

This statement sends units to a semaphore. The format of this statement 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)
```

RCVUNIT

This function receives units from a semaphore and returns the new value of the semaphore. The format of this function 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$token	Token for the semaphore.
units	Number of units to receive from the semaphore.
time$limit	Number of 10-millisecond time units to wait for the units.
stat$var	Variable in which the interpreter returns the status of the receive 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)
```

CRTMBOX

This function creates a mailbox and returns a token for it. The format of this function 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 the type of mailbox to be created. Possible values include: 0 First-in, first-out mailbox 1 Priority mailbox
stat$var	Variable in which the interpreter returns the status of the create operation.

Example:

The following function creates a priority mailbox 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)
```

DELMBOX

This statement deletes a mailbox. The format of this statement 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 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)
```

SENDMSG

This statement sends a message (in the form of an object)to a mailbox. The format of this statement 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$token	Token of the desired response mailbox or semaphore. A zero value indicates that no response is desired.
stat$var	Variable in which the interpreter returns the status of the send operation.

Example:

The following statement sends an object to a mailbox and specifies a response mailbox at which the receive 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)
```

RECVMSG

This function waits for a message (in the form of an object) at a mailbox and returns a token for the object if one is available. The format of this function is as follows:

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

where:

mess$token	Variable in which the interpreter returns a token for the object.
mbox$token	Token for the mailbox.
time$limit	Number of 10-millisecond time units to wait for the object.
resp$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 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)
```

CRTSEGM

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

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

where:

seg$token	Variable in which the interpreter returns a token for the newly created segment.
size	Size in bytes of the segment. A value of zero indicates that a 64K segment is requested.
stat$var	Variable in which the interpreter returns the status of the create 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)
```

DELSEGM

This statement deletes a segment. The format of this statement 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 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)
```

CHAPTER 5. FILES UTILITY SYSTEM

The INTELLEC Microcomputer Development System does not recognize RMX/86 disk files. Consequently you cannot read, write, or format RMX/86 disks directly from the ISIS-II operating system. You can, however, indirectly perform these operations from the Development System by using the RMX/86 Files Utility System.

FUNCTIONS PROVIDED

The RMX/86 Files Utility System is an application system built upon the RMX/86 Operating System which allows you to perform the following operations:

- Formatting an RMX/86 disk.
- Copying a file from an ISIS-II disk to an RMX/86 disk.
- Copying a file from an RMX/86 disk to an ISIS-II disk.
- Deleting a file from an RMX/86 disk.
- Creating a directory on an RMX/86 disk.
- Displaying, on the Development System terminal, the contents of a directory of an RMX/86 disk.

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.

EXECUTING THE UTILITY

Before you can enter commands to the Utility System, you must start it up. This involves putting together some hardware and entering some commands on the INTELLEC Microcomputer Development System terminal.

ASSEMBLING THE HARDWARE

The first thing you must do is to put the hardware together. This involves the following steps.

1) Use the iSBC 957A package to conect the INTELLEC Microcomputer Development System to the iSBC 86/12A board. The procedure for doing this is described in the iSBC 957A INTELLEC--iSBC 86/12A Interface and Execution Package User's Guide.

2) Connect your application system disk drives to their controllers and connect the controllers to the MULTIBUS interface. The procedures for doing this depend upon which controllers you are using. You can find the procedures in the hardware reference manuals for the controllers. The Utility System supports up to two iSBC 204 Shugart drives, two iSBC 204 CDC drives, and two iSBC 206 drives. The interrupt levels and base addresses on the controller boards should be set as follows:

   - For the iSBC 204 controller, set the interrupt level to 5 and the I/O base address to 0A0H.

   - For the iSBC 206 controller, set the interrupt level to 4 and the I/O base address to 068H.

STARTING THE UTILITY

After you have assembled your hardware, place an ISIS-II system diskette containing the iSBC 957A software into drive 0 of your INTELLEC Microcomputer Development System and the Utilities diskette into any other drive. Load the ISIS-II system.

Before going any further, examine the file FILES.CSD on the Utilities diskette. This file is a submit file which loads the Files Utility System from diskette into RAM. As released, it contains the following commands.

```
MON86
L     %ONUCLUS
L     %OIOS
L     %OFILES
L     %OFILES.RJB
E
```

In order for this submit file to function correctly, you must edit this file and make the following change:

- Change the command MON86 so that it reads SBC861.

After you have modified the file, enter the following ISIS-II command:

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

where:

fx	Identifier of the disk 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 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. 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

```
RMX/86 FILES UTILITY Vx.x
```

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

USING THE UTILITY SYSTEM

The Utility System provides seven commands: UPCOPY, DOWNCOPY, DELETE, CREATEDIR, DIR, FORMAT, and EXIT. Each of these is described in detail later in this section. But before you use any of these commands, you should know how the Utility System deals with errors and whether or not you can change disks once the processing has begun.

ERROR MESSAGES

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

```
*** INITIALIZATION ERROR ***
    rr
```

the Utility System is not able to initialize correctly when started. The hardware of your system does not match the software configuration. You might have jumpered boards incorrectly, set the wrong interrupt levels, or left boards unplugged. The rr in this message is a hexadecimal RMX/86 condition code. In order to interpret this code, refer to Appendix B of this manual. The RMX/86 Nucleus, Terminal Handler, and Debugger Reference Manual and the RMX/86 I/O System Reference Manual describe these condition codes in more detail.

If the message is

```
UNRECOGNIZED COMMAND
```

the Utility System does not recognize the spelling of your command. The Utility System prompts for another command.

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

```
ISIS ERROR # nn
```

where nn is in decimal. In order to interpret this error message, refer to the ISIS-II User's Guide. Fatal errors require you to restart the Utility System using the FILES.CSD file.

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

```
RMX EXCEPTION # mm
```

where mm is in hexadecimal. In order to interpret this error message, refer to Appendix B. After this message is displayed, the Utility System prompts for the next command.

CHANGING DISKS

When the Utility System is running and you have already performed an operation on a particular disk, with one exception you cannot remove that disk from the drive and replace it with another. The Utility System is not aware of disk changes, and if you do this, the system treats the second disk as if it were the first, possibly writing over or destroying valuable information. In order to change disks in a drive, you must enter the EXIT command, change the disk, and restart the Utility System by entering the G command.

The one exception to this is the FORMAT command. As described later in this chapter, this command writes RMX/86 formatting information on blank disks. Since the FORMAT command always expects a blank disk, you can replace disks in a drive any number of times if you use only FORMAT. The FORMAT command does not check for a blank disk, however; it destroys all information previously contained on the disk.

COMMANDS

This section lists the commands available with the Utility System and their parameters. Associated with each device name and file name (or path name) parameter is a preface which is a disk identifier. For ISIS-II files, these disk identifiers are the same as those described in the ISIS-II User's Guide. The disk identifiers for RMX/86 devices are similar. They are:

```
:F0:    iSBC 204 Shugart flexible disk drives, units 0 and 1.
:F1:

:F8:    iSBC 204 CDC flexible disk drives, units 0 and 1.
:F9:

:H0:    iSBC 206 hard disk drives, units 0 and 1.
:H1:
```

So, for example, to copy the file JOBA from an ISIS-II disk on unit 1 to a file of the same name on an RMX/86 disk on unit 2 of a Shugart drive, enter the following:

```
UPCOPY :F1:JOBA TO :F2:JOBA
```

UPCOPY

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

```
UPCOPY isis-filename TO rmx-pathname
```

where:

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

DOWNCOPY

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

```
DOWNCOPY rmx-pathname TO isis-filename
```

where:

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

DELETE

This command removes the specified RMX/86 file from storage. The format of this command is as follows:

```
DELETE rmx-pathname
```

where:

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

CREATEDIR

This command creates an RMX/86 directory file. The format of this command is as follows:

```
CREATEDIR rmx-pathname
```

where:

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

DIR

This command lists an RMX/86 directory file at the Development System console. The format of this command is as follows:

```
DIR rmx-pathname
```

where:

rmx-pathname	Path name of the RMX/86 directory file to be listed.

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

```
DIR SIZE: n
```

In this line, n is the total of all the entries ever contained in the directory file. If entries have been deleted, the directory size does not reflect this deletion.

FORMAT

This command writes RMX/86 formatting information on a disk. It destroys all information previously contained on the disk. Each disk must be formatted before it can be used by the RMX/86 Operating System.

FORMAT performs the following operations:

- It records the interleave information on the disk sectors.
- If the named file driver is specified, it records the volume label information on track 0 of the volume.
- If the named file driver is specified, it creates the fnode file, the free space bit map file, and the free fnode bit map file.

The FORMAT command contains parameters which are specified in the form "keyword=value". You can abbreviate each of these keywords as shown. The abbreviations and the format of this command are as follows (brackets [] indicate optional parameters):

```
FORMAT :disk-:id:vol-name [GRANULARITY=gran]
       [INTERLEAVE=ileave][NUMBERFNODES=nodes] [switch]

   or

FORMAT :disk-id:vol-name [GR=gran] [IL-ileave]
       [NF=nodes] [switch]
```

where:

disk-id — Disk identifier which denotes the RMX/86 drive on which the disk resides. Possible values include:

```
F0
F1
F8
F9
H0
H1
```

These values are described earlier in this chapter. Use colons to delimit the disk identifiers.

vol-name — A one to ten character volume name which identifies the disk. ASCII digits, upper and lowercase letters, spaces, and the following special characters can be used in the disk identifier:

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

gran — The granularity, in bytes, for this volume. The granularity is the number of bytes obtained during each disk 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 128 is rounded up to the next multiple of 128.

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 50 is assumed.

switch — A switch which indicates the support option for this volume. One value can be entered for this 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 disk, but does not initialize any of the data structures.

EXIT

This command exits from the Utility System. The format of this command is as follows:

```
EXIT
```

CHAPTER 6. THE RMX/86 DEVELOPMENT PROCESS

In order to produce a final RMX/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 RMX/86-based application system. It is an overview of a typical scenario, illustrating the main points of the development process.

1. Define your application.

2. Do the high level design. This includes:

   - Identify your hardware requirements.

   - Determine which parts of the RMX/86 Operating System parts you need. The configurable nature of the RMX/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 com- pleted the development process, you can remove the Debugger from your system in order to reduce your 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. The Introduction to the RMX/86 Operating System and the RMX/86 Nucleus, Terminal Handler, and Debugger Reference Manual contain information about these processes.

3. Write and debug the task code. As soon as you finish writing each task, you can use either the ICE-86 In-Circuit Emulator or the iSBC 957A package to debug it.

4. Configure the application system. Do this by creating a system configuration file and an individual configur- ation 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. The RMX/86 Configuration Guide for ISIS-II Users describes this process in detail.

5. Assemble your hardware for testing the system.

6. If you are using the I/O System in your application, load the Files Utility System, format your RMX/86 disks, and copy any necessary information to them.

7. Test and debug your system using the Debugger and either the ICE-86 In-Circuit Emulator or the iSBC 957A Interface and Execution package. Continue performing steps 3, 4, and 7 until you are satisfied with your system.

8. Unless you want the Debugger to be a permanent part of your system, perform step 4 again, but omit the Debugger.

9. Burn your debugged code into PROM, and place it on your iSBC 86/12A board.

APPENDIX A. ORIGINAL JUMPERING OF BOARDS

This appendix lists the jumper connections that were made on the iSBC 86/12A board at the factory.

ORIGINAL iSBC 86/12A JUMPERS

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

Jumper Pad	Connection
W1	A-B
W2	A-B
W3	A-B
W4	A-B
W5	A-B
W6	A-B

APPENDIX B. RMX/86 CONDITION CODES

CATEGORY/ MNEMONIC	MEANING	NUMERIC CODE HEX	DECIMAL
Normal			
E$OK	The most recent system call was successful.	0H	0
Exceptional			
Environmental Conditions			
E$TIME	A time limit (possibly a limit of zero time) expired without a task's request being satisfied.	1H	1
E$MEM	There is not sufficient memory available to satisfy a task's request.	2H	2
E$LIMIT	A task attempted an operation which, if it had been successful, would have violated a Nucleus-enforced limit.	4H	4
E$CONTEXT	A system call was issued out of proper context.	5H	5
E$EXIST	A token parameter has a value which is not the token of an existing object.	6H	6
E$STATE	A task attempted an operation which would have caused an impossible transition of a task's state.	7H	7
ENOT-CONFIGURED	The most recently issued system call is not in the present configuration.	8H	8
E$FEXIST	File already exists.	20H	32
E$FNEXIST	File does not exist.	21H	33
E$SUPPORT	Combination of parameters not supported.	23H	35

CATEGORY/ MNEMONIC	MEANING	NUMERIC CODE HEX	DECIMAL
Environmental Conditions (continued)			
E$FACCESS	File access not granted.	26H	38
E$FTYPE	Incompatible file type.	27H	39
E$SPACE	No space left.	29H	41
Programmer Errors			
E$ZERO$-DIVIDE	A task attempted to divide by zero.	8000H	32768
E$OVERFLOW	An overflow interrupt occurred.	8001H	32769
E$TYPE	A token parameter referred to an existing object that is not of the required type.	8002H	32770
E$BOUNDS	A task attempted to access beyond the end of a segment.	8003H	32771
E$PARAM	A parameter which is neither a token nor an offset has an illegal value.	8004H	32772
EBADCALL	Call invoked illegally.	8005H	32773
E$IFDR	Illegal file driver request.	8020H	32800
E$NOUSER	No default user.	8021H	32801
E$NOPREFIX	No default prefix.	8022H	32802
ENOTDEV$-NAME	Not a valid device name.	804DH	32845
ENOT-CONN$NAME	Not a valid connection name.	804EH	32846
Asynchronous Conditions			
E$MEM	There is not sufficient memory available to satisfy a task's request.	2H	2

CATEGORY/ MNEMONIC	MEANING	NUMERIC CODE HEX	DECIMAL
Asynchronous Conditions (continued)			
E$LIMIT	A task attempted an operation which, if it had been successful, would have violated a Nucleus-enforced limit.	4H	4
E$CONTEXT	A system call was issued out of proper context.	5H	5
E$FEXIST	File already exists.	20H	32
E$FNEXIST	File does not exist.	21H	33
E$DEVFD	Device and file driver are incompatible.	22H	34
E$SUPPORT	Combination of parameters not supported.	23H	35
E$EMPTY$-ENTRY	Empty directory entry.	24H	36
EDIREND	End of directory.	25H	37
E$FACCESS	File access not granted.	26H	38
E$FTYPE	Incompatible file type.	27H	39
E$SHARE	Improper file sharing requested.	28H	40
E$SPACE	No space left.	29H	41
E$IDDR	Illegal device driver request.	2AH	42
E$IO	I/O Error.	2BH	43
E$FLUSHING	Connection is flushing requests.	2CH	44
E$IUVOL	Illegally named volume.	2DH	45

INDEX

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. 1040 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
Attn: Technical Publications
3065 Bowers Avenue
Santa Clara, CA 95051