intel®

Guide to the iRMX® I Interactive Configuration Utility

Order Number: 462923-001

intel®

Guide to the iRMX® I Interactive Configuration Utility

Order Number: 462923-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

Above	iLBX	iPSC	Plug-A-Bubble
BITBUS	i$_m$	iRMX	PROMPT
COMMputer	iMDDX	iSBC	Promware
CREDIT	iMMX	iSBX	QUEST
Data Pipeline	Insite	iSDM	QueX
Genius	int$_e$l	iSSB	Ripplemode
î	Intel376	iSXM	RMX/80
i	Intel386	Library Manager	RUPI
I^{2}ICE	int$_e$lBOS	MCS	Seamless
ICE	Intelevision	Megachassis	SLD
iCEL	int$_e$ligent Identifier	MICROMAINFRAME	UPI
iCS	int$_e$ligent Programming	MULTIBUS	VLSiCEL
iDBP	Intellec	MULTICHANNEL	376
iDIS	Intellink	MULTIMODULE	386
	iOSP	OpenNET	386SX
	iPDS	ONCE	
	iPSB		

REV.	REVISION HISTORY	DATE
-001	Original Issue.	03/89

PREFACE

INTRODUCTION

This manual describes the Interactive Configuration Utility (ICU) and explains its use. It does not explain each screen displayed by the ICU. For a description of the ICU screens and their parameters, refer to the *iRMX® I Interactive Configuration Utility Reference Manual.*

READER LEVEL

The manual assumes that you are familiar with the monitor and keyboard from which you run the ICU. It is also helpful if you are familiar with the following:

- The iRMX® I Operating System
- PL/M-86
- LINK86 and LOC86

MANUAL OVERVIEW

This manual is organized as follows:

Chapter 1	This chapter provides introductory material to configuring an iRMX I system using the Interactive Configuration Utility (ICU).
Chapter 2	This chapter describes how to generate a system.
Chapter 3	This chapter describes how to prepare application jobs.
Chapter 4	This chapter provides overview information on how to add users to your system. For detailed information on adding users, refer to the *Operator's Guide to the iRMX® Human Interface*.
Chapter 5	This chapter describes how to load and test the system.
Appendix A	This appendix lists files created by the ICU.
Appendix B	This appendix shows an example configuration session.
Appendix C	This appendix describes how to program a generated 286-based system into PROM devices.

CONVENTIONS

This manual uses the following conventions:

- Information appearing as UPPERCASE characters when shown in keyboard examples must be entered or coded exactly as shown. You may, however, mix lower and uppercase characters when entering the text.
- Fields appearing as lowercase characters within angle brackets (< >) when shown in keyboard examples indicate variable information. You must enter an appropriate value or symbol for variable fields.
- User input appears in one of the following forms:

```
as blue text
```

```
as bold text within a screen
```

- All numbers unless otherwise stated are assumed to be decimal. Hexadecimal numbers include the "H" radix character (for example, 0FFH).

CONTENTS

Chapter 3. Preparing Application Jobs (continued)

Tables

Figures

Figures (continued)

INTRODUCTION TO CONFIGURATION 1

1.1 INTRODUCTION

The iRMX® I Operating System is modular in structure, enabling you to include or omit subsystems according to your needs. It is also compatible with a variety of peripheral boards. The Interactive Configuration Utility (ICU) is designed to help you take advantage of this flexibility.

This chapter provides an overview of the ICU. It explains the configuration process, configuration environment, ICU files, ICU commands, error messages, the utilities that comprise the ICU, and more. The chapter includes many important details for using the ICU. Intel recommends that you read this chapter carefully before attempting to configure your system.

1.2 WHAT IS CONFIGURATION

Configuration is the process of selecting your application's hardware and operating system layers and then linking and locating the entire operating system. The tool used for configuration is the Interactive Configuration Utility (ICU). The ICU is a menu-driven utility which presents a series of screens that prompt you for information. The information is stored in a definition file that is then used to generate the new system.

The objective of running the ICU is to build a definition file that contains all of the configuration information. This file contains two kinds of information:

- Initialization parameters
- A set of variables specifying which operating system layers and device drivers are to be linked together with your application software

Intel provides six definition files you can use as a starting point. If you run the UPDEF86 Utility supplied with this release, you can also use a definition file from iRMX 86, Release 7. As you perform the configuration process, you alter the chosen definition file to match your target system.

1.3 WHEN TO USE THE ICU

You should use the ICU whenever one of the following is true:

- You want to generate the configuration files that describe your system.
- You are using a system other than one described by an Intel-supplied definition file.
- You are changing an existing system's hardware and/or software (for example, adding a new disk drive).

1.4 ICU LOCATION

The ICU files are located in the directory :SD:RMX86/ICU (see Figure 1-1). When working with the ICU, you must use the full pathname in each command or create an alias for the pathname.

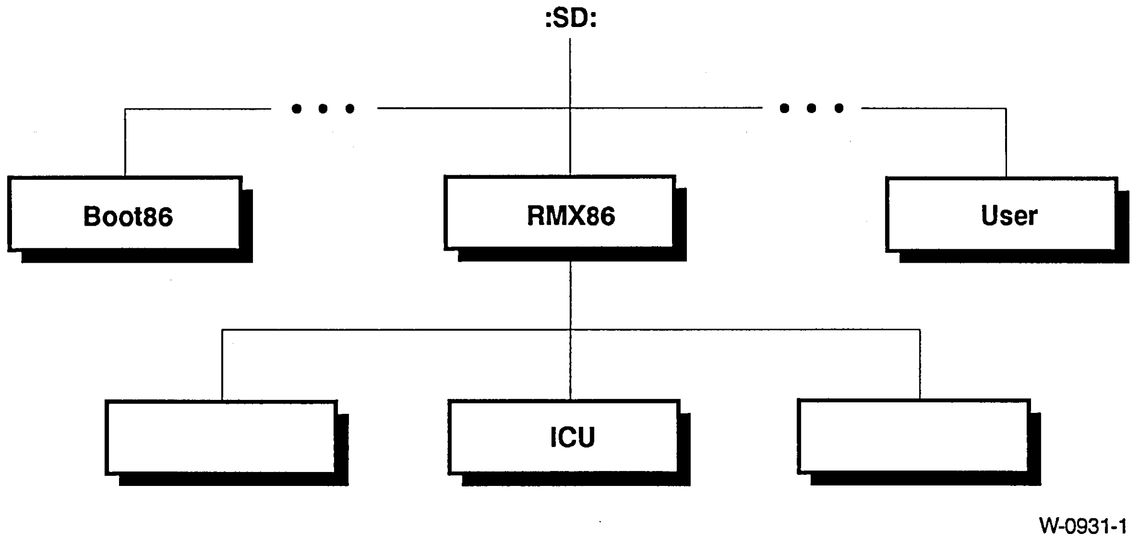


Figure 1-1. Location of the ICU Directory in an Intel-supplied System

1.5 THE GENERAL PROCESS OF USING THE ICU

You configure a system in three steps:

1. Interactively modify a definition file (see Figure 1-2). To do this, invoke the ICU and then supply information to fill in screens that the ICU presents. (This step can be omitted if your system matches one of the Intel-supplied definition files.)
2. When you finish configuring the operating system, use the ICU to generate new configuration files as defined in your modified definition file (see Figure 1-3). The end product is a group of files that define the system.
3. Exit from the ICU, and at the Human Interface level, execute the submit file created by the ICU during the generate step (see Step 2). This creates the new version of the operating system which can then be loaded and executed.

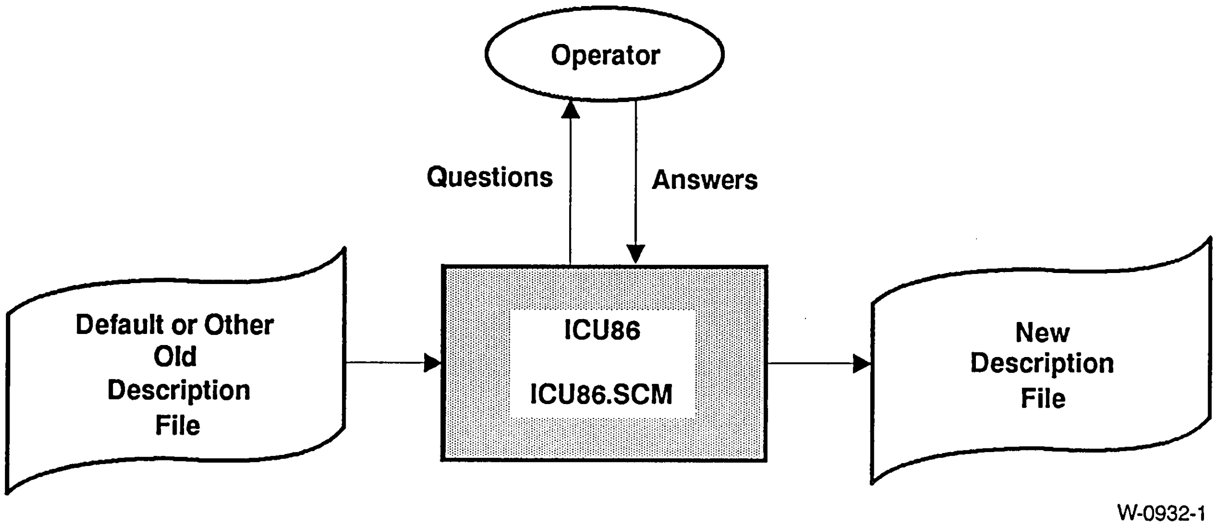


Figure 1-2. First Step: Editing a Definition File

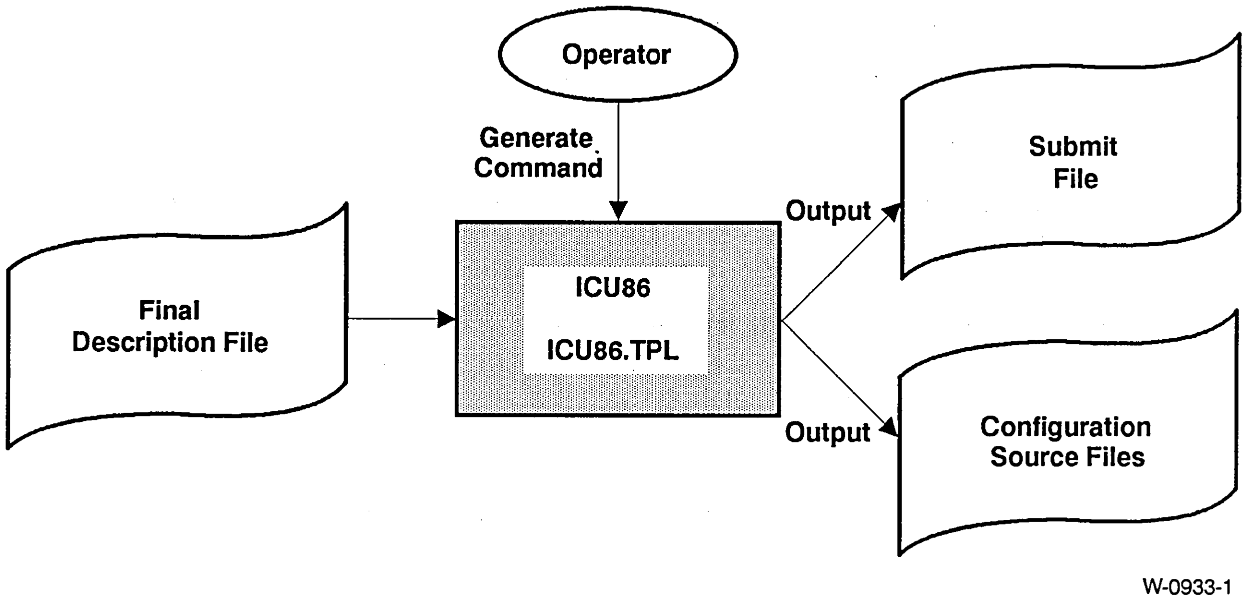


Figure 1-3. Second Step: Generating a Submit File and Source Files

1.6 WHAT TO DO BEFORE INVOKING THE ICU

Before you invoke the ICU, you must perform checks on your existing system and make several decisions. The following sections provide the information you need to know before invoking the ICU.

1.6.1 Configuration Environment

You can run the ICU on an iRMX I-based system with 1 Mbyte of RAM memory (which allows a user partition of 384K bytes), a hard disk, and the iRMX I Operating System. For information on changing the amount of memory allocated to a user, see the section on editing the :CONFIG:USER/username file in the *Operator's Guide to the iRMX® Human Interface*.

1.6.2 Ensuring the ICU Files are on Your Hard Disk

Contained on the iRMX I Operating System release diskettes or tape are the files to run the ICU. These files must be on the hard disk before you can invoke the ICU. Follow the instructions in the *iRMX® I Hardware and Software Installation Guide* to copy these files to your system.

Table 1-1 lists all of the files required to run the Interactive Configuration Utility for iRMX I systems and for iNDX systems. Check that your hard disk contains all of the files required by your system. If your hard disk does not contain the required files, return to the instructions in the *iRMX® I Hardware and Software Installation Guide*. Following the directions in that manual will place all the iRMX I files into the standard directory structure.

Table 1-1. iRMX® I ICU Files

Function	Filename
ICU86 - first stage	ICU86
ICU86 - second stage	ICU86.862
Screen Master File	ICU86.SCM
Template File for System Generation	ICU86.TPL
Update Definition Utility	UPDEF86
User Device Support Utility	UDS86
UDS Screen Master File	UDS86.SCM
Template example - (minimum UDS input file)	TEMPLATE_1.UDS
Template example - (UDS input file containing user help built into help text)	TEMPLATE_2.UDS
ICUMRG86 Utility	ICUMRG86
Files containing registration information that is displayed when you invoke the ICU.	REGISTER.ICU REGISTER.MSG
Definition file for an iRMX I Multi-User System designed to run on 8086-based micro-computers	8635.DEF
Definition file for an iRMX I Multi-User System designed to run on the iSBC® 186/03A board	18603a.DEF
Definition file for an iRMX I Multi-User System designed to run on the iSBC 186/51 board	18651.DEF
Definition file for an iRMX I Multi-User System designed to run on the iSBC 186/48 board	18848.DEF
Definition File for an iRMX I Multi-User System designed to run on the iSBC 286/10(A) and iSBC 286/12 boards	28612.DEF
Definition File for an iRMX I Multi-User System designed to run on the iSBC 386/2X and iSBC 386/3X boards	38620.DEF
Definition file used if the input-file specified during invocation does not exist.	ICU86.DEF

1.6.3 Choosing Your Definition File

If you have never configured an iRMX-based system before, you should choose one of the Intel-supplied, multi-user definition files (listed in Table 1-1) as input into the ICU. You can build a definition file screen by screen, but you will save time by starting with a standard definition file. Details on the standard definition files are given in the *iRMX® I Hardware and Software Installation Guide.*

If you created a definition file using an iRMX 86 Release 7.0 version of the ICU, you can use this file as input to the iRMX I.8 ICU only after running the UPDEF86 Utility. Once you create an iRMX I.8 definition file, you cannot use it as input into the iRMX 86 Release 7.0 ICU.

The Intel-supplied definition files define 8086-, 80186-, 80188-, 80286- and 386™-based MULTIBUS I systems. These systems are fully configured multi-user iRMX I Operating Systems. Each layer implements all its system calls and most of the features and drivers provided by the iRMX I Operating System. Multiple users can communicate with the operating system interactively through a terminal or via an application program, and can access secondary storage. The definition files include UDI so that you can run languages, such as PL/M-86, PASCAL, and FORTRAN.

To define your own system, modify the definition file that comes closest to your needs. To see the contents of a definition file, use the LIST command (described later in this chapter). Details of board configuration and interrupt levels are given in the *iRMX® I Hardware and Software Installation Guide.*

1.6.4 Checking Access Rights to Definition Files

If you use the ICU on an iRMX I-based system, the operating system makes sure you have the proper access to both the definition files and their respective directories. This check is performed in two instances: when you invoke the ICU and when you enter the G (generate) command (discussed in a later section). If you do not have proper access, one or more of the following situations can occur:

- The ICU will be unable to read the input definition file.
- The ICU will be unable to save the changes you make during the ICU session.
- The ICU will be unable to create the generation files necessary to complete the configuration process.

Following the installation instructions in the *iRMX® I Hardware and Software Installation Guide* ensures that the user WORLD can generate a new version of the operating system.

To check your access rights to directories and files contained in directories, use the DIR command followed by the E[xtended] or L[ong] options. For example, you can check your access rights to the :SD:RMX86/ICU directory and the 28612.DEF file by using the following commands:

```
- DIR :SD:RMX86 E[xtended] <CR>
- DIR :SD:RMX86/ICU E[xtended] <CR>
```

For more information on using the Human Interface commands, see the *Operator's Guide to the iRMX® Human Interface*.

The access rights needed to use the ICU successfully vary according to the operations to be performed. In all cases where the G (generate) command is to be used, you must have Add Entry access to the directory containing the definition file and to the directories you specify in the "Generate File Names" screen. In other cases, the access required depends on the kind of file (new or existing) and whether it is an input or output file. The following paragraphs describe the access rights required in different circumstances.

- If you specify an existing definition file as an input file, you must have Read access to the definition file.
- If you specify an existing definition file as an output file, you must have Delete and Write access to the definition file. You must also have Add Entry access to the directory containing the definition file.
- If you specify an existing definition file as the only definition file on the command line, the file serves as both an input and an output file. In this case, you must have Read/Write and Delete access to the definition file. (Refer to the *iRMX® I Interactive Configuration Utility Reference Manual* for more information about the "Generate File Names" screen.)
- If you specify a new file as the only file on the command line, the file serves as both an input file and an output file. In this case, you must have Add Entry access to the directory containing the new file.

If you do not have the correct access rights, the ICU returns an E$FILE_ACCESS error message. Additional information about access rights can be found in the *iRMX® Extended I/O System User's Guide* and the *Operator's Guide to the iRMX® Human Interface*.

EXAMPLES:

To use the G command without causing an error, you must have Add/Delete Entry access to the directory in which you are working. This section describes the various ways of invoking the ICU and the access rights required.

```
    - ICU86 :home:new.def  <CR>
```

where:

:HOME:NEW.DEF — Pathname of a new definition file. The ICU uses this file as both the input file and the output file.

You must have Add Entry access to the directory containing NEW.DEF (:HOME:). If you do not have Add Entry access, the ICU returns the following message:

```
*** I/O Error in file: :HOME:NEW.DEF
0026: E$FILE_ACCESS
```

If the directory does not exist, the ICU returns the following message:

```
*** I/O Error in file: :HOME:NEW.DEF
0021: E$FILE_NOT_EXIST
```

```
    - ICU86 dir/old.def  <CR>
```

where:

DIR/OLD.DEF — Pathname of an existing definition file. The ICU uses this file as both an input and an output file. It can be a new or existing file.

You must have Read/Write and Delete access to save changes to OLD.DEF.

```
    - ICU86 /rmx86/icu/input.def TO dir/output.def  <CR>
```

where:

/RMX86/ICU/INPUT.DEF	Pathname of a standard definition file. The ICU uses this file as the input file.
DIR/OUTPUT.DEF	Pathname of the output definition file. This file can be a new or existing file. You must have Add Entry access to the directory containing OUTPUT.DEF (DIR). In addition, if OUTPUT.DEF is an existing file, you must have Delete access to it.

1.7 DISTINGUISHING ICU-GENERATED FILES

Each time you generate your system, the ICU generates a set of ICU files. To help you distinguish your generation files from each other and to determine which input definition file generated the ICU files, you can use one of these options:

- Create a new directory to contain your definition file.
- Use the prefix option supplied by the ICU.

The following sections describe each method in more detail.

1.7.1 Creating Directories for iRMX® I-Based Systems

Intel recommends that you maintain the default directory structure by placing any new system definition files in a new directory nested in your :HOME: directory. Before invoking the ICU, you should create a copy of the input definition file in your working directory to avoid corrupting the original file. The following example illustrates how to do this using the 28612.DEF file as the starting definition file:

```
    - CREATEDIR 28612 <CR>
    - ATTACHFILE 28612 <CR>
```

Once you attach the new directory (28612) as the working directory, invoke the ICU. For example

```
    - ICU86 /rmx86/icu/28612.def TO new.def <CR>
```

A convenient convention to use is to create the working directory with the same name as the definition file (without the .DEF extension). The operating system produced should have the same name as the definition file (with no extension).

1.7.2 Using the Prefix Option

A second method to distinguish your ICU generated files from each other is to use the prefix option supplied by the ICU. You can select the prefix option when entering the Generate (G) command. The ICU then displays a prompt (see Chapter 2 for the actual screen) asking you for the prefix letter you wish to assign to the files created by the ICU. For example, if you choose the letter "Q" as your prefix option, a "Q" will precede all the files generated when you enter the Generate command on the menu screen. In this case, the files generated for the Nucleus will be

```
QNTABL.A86
QNDEVC.A86
QNUC.CF
```

The files created for all other subsystems will also be preceded by the letter "Q" as in the example above. If you want to generate configuration files for more than one system, choose a different prefix option each time. Intel also recommends that your input definition file start with the same prefix letter you assign to the generated files. This allows you to easily determine which definition files created each set of output files. If you create a file with a prefix that already exists, the ICU overwrites the previous file.

If you do not want to use the prefix option, enter a carriage return when you are prompted for the prefix. This causes the ICU to generate the output files without a prefix (see Appendix A for a complete list of the ICU generated files).

1.8 INVOKING THE ICU

This section lists the syntax necessary to invoke the ICU. When invoking the ICU, be sure that the ICU86.SCM file and the ICU86.TPL file reside in the same directory as the ICU86 file that you are invoking.

The syntax for invoking the ICU is as follows (brackets indicate optional items):

```
ICU86 [input-file-name TO] output-file-name
```

where

input-file-name	Name of the definition file from which the ICU obtains configuration information. This is typically an Intel-supplied definition file or a definition file from a previous use of the ICU.
output-file-name	Name of the definition file to which the ICU writes updated configuration information.

The following guidelines must be followed when specifying input and output files:

input-file-name: If both an input-file-name and an output-file-name are specified, the input-file-name must represent an existing definition file created by the ICU.

output-file-name only: If the input-file-name is omitted, you enter only the output-file-name, the ICU uses the output-file-name as both the input and output definition file. When the ICU session is complete, the ICU writes the updated configuration information back to the output-file-name.

The output-file-name entered can represent a new or an existing definition file. If the output-file-name specified does not exist, the ICU searches the directory containing ICU for a file named ICU86.DEF. If ICU86.DEF exists, the ICU uses it as the input file. In any case where an input file does not exist, the ICU displays the following message among the main screen display:

```
NEW CONFIGURATION FILE
```

and the session starts with the ICU default values. After saving or exiting, the edited file is stored in the named output file.

1.8.1 Invocation Error Messages

When issuing the invocation previously described, a number of error messages can occur. These error messages are described in the following paragraphs.

Invoking the ICU with no parameters or invalid parameters causes the following message to be displayed:

```
*** INVALID INVOCATION ***
USAGE:  ICU86  [input-file TO] output-file
```

Invoking the ICU with a corrupted definition file, or a file that is not a definition file, causes the following message to be displayed:

```
*** ERROR - FILE: <file_name> IS NOT VALID
```

On invocation the ICU validates the file version numbers. ICU86.SCM, ICU86.TPL, and each definition file have an Intel Version Number, an Update Version Number, and a User Version Number. The Intel version number changes whenever Intel upgrades the ICU to support a new release. The Update version number changes whenever the ICU is upgraded, using the ICUMRG86 utility, to support an update; and the User version number changes whenever a user device is added (using the UDS86 and ICUMRG86 utilities).

The ICU checks the version numbers when it is invoked. If there is an inconsistency in the version numbers of either ICU86.SCM or ICU86.TPL, the ICU displays the following error message and returns control to the operating system:

```
*** ERROR - INCONSISTENCY IN THE VERSION OF THE INTERNAL ICU FILES
    Versions:          Intel   Update      User
            ICU86.SCM  <Intel> + <Update> <User version>
            ICU86.TPL  <Intel> + <Update> <User version>
```

If the inconsistency is in the definition file, the ICU displays the following warning message and asks for permission to update the file or restore from the file. (The update process is a simpler, faster operation than restore and requires no additional user input.)

```
*** WARNING - DEFINITION FILE VERSION IS NOT CORRECT.

                                Intel   Update  User
            ITS VERSION IS:    <the inconsistency>
            VERSION EXPECTED:  <correct version>
```

If the ICU needs to restore from a file in order to use it, you will be prompted as follows:

```
Do you want to restore from the file? y/[n]
```

A response of "No" causes the ICU to stop executing. A "Yes" response means the ICU will restore the backup information stored in the definition file (discussed later in this chapter). If the Update version number is higher than the ICU version number, you are probably using the wrong version of the ICU. In this case, the ICU displays this warning before the restore prompt:

```
*** WARNING - The Definition File version is NEWER
```

However, if the ICU is able to use the file without restoring, it prompts with

```
Do you want to update the file? y/[n]
```

A response of "No" causes the ICU to stop executing. A "Yes" response causes the ICU to update the file. Since the ICU processes definition files with inconsistent version numbers, you can use all of the Intel-supplied definition files as input for your own tailor-made ICU.

The ICU issues the restore prompt if it discovers any of the following in the definition file:

- Inconsistency in the Intel version number.
- Inconsistency in the User version number, if the file contains user devices added with UDS86 and ICUMRG86.
- Inconsistency in the Update version number, if the file version number is higher (newer) than the ICU version number.

The ICU prompts for permission to use the file without restoring in the following cases:

- Inconsistency in the User version number, if the file does not contain user devices.
- Inconsistency in the Update version number, if the file version number is smaller than the ICU version number.

1.9 WHAT TO DO AFTER INVOKING THE ICU

After you invoke the ICU, the following registration message screen is displayed:

```
iRMX I Interactive Configuration Utility For iRMX I,<version>
Copyright <years> Intel Corporation - All Rights Reserved

               *-*-*  PLEASE REGISTER THIS SOFTWARE  *-*-*

By registering your software, you will receive free initial software
support, including updates to the software when available, telephone
support for problems you may encounter, technical literature, membership
in a users' software library, and more.

To register, either MAIL the registration card packaged with this product
or CONTACT Intel using one of the numbers listed below.  Please see the
registration card for additional phone numbers.  When you register the
software, you will receive a registration number that will erase this
message.

USA          800-INTEL-4-U (USA only)   Italia       (02) 8 24 40 71
             602-869-4001 (FAX)         Nederlanden  (010) 4 21 23 77
UK           (0793) 641469              Israel       (03) 498080
Deutschland  (0 89) 9 03 20 25          Nippon       029747-8511
France       (01) 30 64 56 95           Hong Kong    852-5-844-8575
Sverige      (08) 7 34 01 00

Enter your registration number and <CR>, or <CR> for the main menu:
```

The string <version> represents the ICU version number. The string <years> represents the copyrighted years of the product.

The registration screen appears each time you invoke the ICU until you obtain your registration number and enter it at the prompt line at the bottom of the screen.

When you press <CR>, the main menu screen is displayed as follows:

```
    For general help in any screen enter H <CR>.

    The following commands are available

    Change
    Generate
    List
    Save
    Quit
    Exit
    Replace
    Detail-Level
    Backup

     ENTER COMMAND :
```

The screen shown above contains the main ICU menu. Whenever you see this screen you are in command mode.

Whenever you are in command mode, you must enter one of the commands listed or an "H" for help. All of your responses should be followed by a carriage return. The ICU regards all invalid input as a response of "H <CR>" and displays the "Help" screen until a valid response is entered.

The following sections describe the choices on the menu screen.

1.9.1 Help Command

The Help command displays information about the ICU commands available to you. The syntax of the Help command is as follows:

```
H <CR>
```

If you enter H (help) and a carriage return, the ICU will display the following screen:

```
The Change (C) command allows you to specify the configuration
parameters that define your system.  To get to a specific
screen, type 'C screen$name'.  C? gives you a list of all the
screen names.

The Generate (G) command creates the submit file and all PLM
and assembler files required to create your iRMX I
system.

The List (L) command shows you the current state of your
configuration file.  To copy the values that define your
system to a file, type 'L file$name'.

The Quit (Q) command leaves the ICU without saving any
changes.

The Exit (E) command leaves the ICU and saves all the
changes.

The Save (S) command saves all the changes without leaving
the ICU.

The Replace (R) command replaces the current control
character.

The Detail level (D) command sets the level of detail.

The Backup (B) command writes a backup file.

TYPE <CR> to Continue
```

1.9.2 Change Command

The Change command enables you to begin editing the definition file. The syntax of the Change command is as follows:

```
C[hange] [Screen Abbrev] <CR>
              or
      C[hange] ? <CR>
```

where:

C or Change	Starts editing the definition file from the first screen. The first time you run the ICU you should use this option.
screen abbrev	Begins editing at a specific screen. For example, if you enter "C", a space, the abbreviation of an existing screen, and a carriage return, the ICU enables you to start editing your definition file from that particular screen. If you enter a screen abbreviation incorrectly, the ICU displays a screen containing all the screen names and abbreviations (see Table 1-2). The abbreviation enclosed in parentheses indicates what must be entered for each screen.
?	Causes the ICU to display a screen with all the screen names and abbreviations.

Table 1-2 lists all the possible screen names. The screens are displayed in order from left to right, that is the "Interrupts" screen is displayed after the "Hardware" screen. Device drivers are listed at the end of the table.

If you did not invoke the ICU with the name of an existing definition file, you should start your edit with the "Hardware" screen. If you did invoke the ICU with the name of an existing definition file, you can start your edit with the name of any screen that the input definition file has already defined. If you enter a valid screen name but that screen is not configured into your definition file, the ICU displays the next "main" screen followed by this warning:

```
***Warning - The screen requested cannot be displayed
```

The ICU progresses from screen to screen in a logical order. Refer to Figure 1-4 for the logical flow of the ICU.

Table 1-2. Screen Names

MAIN SCREENS					
(HARD)	Hardware	(INT)	Interrupts	(SLAVE)	Slave Interrupt
(A186)	80186	(MBII)	MBII Hardware	(RAMEM)	RAM Memory
(ROMEM)	ROM Memory	(SUB)	Sub-systems	(HI)	Human Interface
(HIJOB)	HI Jobs	(RES)	Resident User	(PREF)	Prefixes
(HILOG)	HI Logical	(APPL)	Application Loader	(REM)	Remote file Access
(REMFS)	Remote Server	(EIOS)	EIOS	(ESCS)	EIOS Sys Calls
(ABDR)	Auto Boot Dev	(LOGN)	Logical Names	(IOUS)	I/O Users
(IOJOB)	I/O Jobs	(BIOS)	BIOS	(NOFSC)	Non-file SCalls
(PFSC)	Phys file SCalls	(SFSC)	Stream file SCalls	(NFSC)	Named file SCalls
(IDEVS)	Intel Devices	(USERD)	User Devices	(UDDM)	UDS Dev Drvr Mods
(SDB)	System Debugger	(THDDB)	Dynamic Debugger	(NUC)	Nucleus
(OBSC)	Object SCalls	(JTSC)	Job & Task SCalls	(ECGSC)	Exchange SCalls
(FSSC)	Free Space SCalls	(INTSC)	Interrupt SCalls	(EXTSC)	Extension SCalls
(EXCSC)	Exception SCalls	(USERJ)	User Jobs	(USERM)	User Modules
(ROM)	ROM Code	(INCL)	Includes & Libs	(GEN)	Gen. File Names
(COMNT)	Comments screen				

DEVICE DRIVER SCREENS			
(D214)	Mass Storage Controller	(D350)	Line Printer - iSBX 350
(D286)	Line Printer - iSBC 286/10	(D8848)	Terminal Comm controller
(D8251)	8251A Terminal	(D2530)	82530 Terminal
(D534)	534 Terminal	(D544)	544A Terminal
(D8274)	8274 Terminal	(DTHD)	Terminal Handler
(D220)	iSBC 220	(D218)	iSBX 218A
(D208)	iSBX 208	(D254)	iSBC 254
(DRAM)	RAM Disk	(D264)	iSBC 264
(DSCSI)	SCSI		

1.9.3 Generate Command

The Generate command creates all the ASM, PL/M, and submit files required to configure the iRMX I system. The syntax of the Generate command is as follows:

```
G[enerate] <CR>
```

Refer to Chapter 2 for more information on generating a system.

1.9.4 List Command

The List command enables you to list the contents of your definition file to a file or to a device. This command lists the contents of those screens that you selected to define your system. The syntax of the List command is as follows:

```
L[ist] [name] <CR>
```

where:

L or List	Lists the contents of your screens.
name	Specifies an iNDX or iRMX I device or file. If you omit the name, the terminal (:CO:) is assumed. You should list the definition file to a filename rather than to the terminal since the display scrolls rapidly. If you want to use your terminal to review your definition file, use the Change command to view just those screens you want.

After the ICU lists the definition file to the specified filename, it notifies you that the definition file has been listed and returns to command mode. For example, if you listed the ICU screens to a file called ICU86.LST, the ICU would display

```
The Definition File has been listed to file: ICU86.LST
```

followed by the main menu screen.

1.9.5 Save Command

The Save command updates your definition file with all of the changes you entered during the current ICU session. The syntax of the Save command is as follows:

```
S[ave] [name] <CR>
```

where:

S or Save	Saves all the changes made in this session.
pathname	Pathname of a file to use instead of the default output-file-name to save changes to the definition file.

When the Save command is entered (followed by a carriage return), the ICU updates the file you specified as the output-file-name. After the ICU updates the definition file, it notifies you that the specified file has been updated and returns to command mode. For example, if you invoked the ICU using 28612.DEF as the output-file-name, the ICU would display this message followed by the menu screen:

```
The Definition File has been written to file:  28612.DEF
```

To be sure you are updating the right file, use the List command before you save your definition file. The List command displays the name of the output definition file at the top of each ICU screen.

1.9.6 Quit Command

The Quit command enables you to stop your current ICU session without updating the definition file. The syntax of the Quit command is as follows:

```
Q[uit] <CR>
```

After you enter the Quit command (followed by a carriage return), the ICU may display the prompt "Do you want to quit without saving your changes? y/[n]" to ensure that you did not accidentally enter the Quit command. Your response to this prompt should be either "Yes" or "No". The ICU only displays this prompt if you use the Quit command after making changes to an existing definition file or creating a new definition file. If no changes were made to the definition file before the Quit command was entered, no prompt is displayed.

1.9.7 Exit Command

The Exit command exits the ICU and updates the definition file with all of the changes from the current ICU session. The syntax of the Exit command is as follows:

```
E[xit] [pathname] <CR>
```

where:

E or Exit	Exits the ICU saving all the changes made in this session.
pathname	Pathname of a file to use instead of the output-file-name to save changes made to the definition file.

You should always use either the Exit or Save command after using the Generate command.

1.9.8 Replace Command

The Replace command enables you to change the control character that the ICU uses in the special editing commands. The control character precedes special editing commands. The syntax of the Replace command is as follows:

```
R[eplace] <CR>
```

The default control character is the caret (^). If your terminal does not support this character, or you prefer a different character, use the Replace command to change it to any character of your choice.

After you enter the Replace command (followed by a carriage return), the ICU displays the following screen:

```
Input new control character :
```

Enter the new control character you select, followed by a carriage return.

1.9.9 Detail-Level Command

The Detail-Level command enables you to set the level of detail you want when displaying the ICU screens. The syntax of the Detail-Level command is as follows:

```
D[etail-Level] <CR>
```

This command provides the option of selective screen displays. Rather than viewing all the screens, you can elect to see only screens of a particular type. There are four possible levels you may request:

All	Shows all the screens
Devices	Shows only device screens
Operating System	Shows all non-hardware related screens
Jobs	Shows only the job screens (such as User, I/O and User Modules screen)

After you enter the Detail-Level command (followed by a carriage return), the ICU displays the following:

```
The following levels of detail are available:

All
Devices
Operating-System
Jobs

ENTER Level of Detail :
```

If you enter an invalid response, the ICU redisplays this screen until it receives a valid response.

1.9.10 Backup Command

The Backup command writes an ASCII backup file containing a list of all the parameter abbreviations and their current values. The syntax of the Backup command is as follows:

```
B[ackup] filename <CR>
```

where:

B or Backup	Writes a backup file.
filename	Name of the file that will contain the backup information.

The backup file is used as input to the ICU during the restore process (discussed later in this chapter). Remember, the information in the backup file is part of the definition file. The advantage of creating a backup file is that it is in ASCII which is easier and safer to use with other utilities or electronic mail.

When the backup command has completed, the ICU displays a message that the filename specified has been backed-up. It then returns to command mode. For example, if you backed-up your definition file to a file named UPDATE.BCK, the following screen would be displayed.

```
 The Definition File has been backed-up to file: UPDATE.BCK


 For general help in any screen enter H <CR>.

The following commands are available

Change
Generate
List
Save
Quit
Exit
Replace
Detail-Level
Backup

 ENTER COMMAND :
```

1.9.11 Aborting ICU Commands

The ICU enables you to abort an ICU process, without losing any information, by entering CONTROL-C. If you enter CONTROL-C during the execution of an ICU command (Generate, List, etc.) the ICU stops executing the current command, and returns to the main menu screen. The ICU handles CONTROL-C differently for each command.

- If entered in command mode, or during SAVE, QUIT, EXIT or BACKUP, it is ignored.
- If entered during CHANGE, it displays the following message:

  ```
  'Type C to EXIT to the Main-Menu'
  ```

- If entered during GENERATE, the ICU finishes writing the file being generated, displays

  ```
  '*** Process ABORTED.'
  ```

  and returns to command mode.
- If entered during LIST, the ICU displays the Process ABORTED message, writes the message to the file or device specified in the List command, and returns to command mode.
- If entered during REPLACE or DETAIL-LEVEL, the ICU returns to command mode.
- If entered while restoring, the ICU displays the following message and returns control to the operating system.

  ```
  '***  Process ABORTED - The Definition File was not restored.'
  ```

1.10 CHANGING A DEFINITION FILE

It is possible to change a definition file by entering the "Change" command on the menu screen. The ICU then shows one screen of information at a time. Each screen pertains to a specific area of configuration. The information displayed on the screen consists of a series of prompts and default values. Any of the default values can be changed. However, the changes you make are not immediately displayed on the screen. They are displayed only when you reshow the screen using the editing command ^R (or R) (discussed later in this chapter) or just a carriage return.

Entering another carriage return after using one to reshow a screen causes you to proceed to the next screen. The changes are recorded in the definition file when you exit the ICU using the "E" command or when you enter the "S" command while still in the ICU.

1.10.1 Explanation of the Basic Screen Elements

The following definitions will help you understand the various parts of a screen. The following screen illustrates the defined terms.

```
(SCABV)   SCREEN NAME

(ABV) PARAMETER DEFINITION [range of values]      XXX
(ABV) PARAMETER DEFINITION [range of values]      XXX

                          •
                          •
                          •


Enter [Abbreviation = new value / Abbreviation ? / H ] :
<prompt line>
```

(SCABV)	The abbreviation enclosed in parentheses identifies the screen being displayed. This abbreviation is used with the "Change" or "Find" commands (discussed later in this chapter) to access a screen.
SCREEN NAME	The name of the screen.
(ABV)	The abbreviation enclosed in parentheses identifies the parameter whose existing value can be replaced.
PARAMETER DEFINITION	This definition briefly describes the parameter that you can change.
[range of values]	This defines the range of acceptable values for this parameter.
XXX	The value in the current definition file. If the existing value is not what you want, replace it with any other value within the range of values.
<prompt line>	This line is where you enter changes to the screen. The cursor is located at the beginning of this line ready for you to enter one of the following: • An abbreviation, an equal sign (=) and a new value • An abbreviation and a "?", if you need an explanation of the parameter

- A ^H (H), if you need general help in understanding the screen types or editing commands
- A "?", if you need an explanation of the specific screen

Data you enter on the prompt line should be followed by a carriage return (<CR>).

The following screen shows all of the features described above. The screen abbreviation is (HARD) and the screen name is "Hardware". There are 16 parameter lines and a prompt line. Each parameter line includes a range of legal values which may be entered if the default value does not meet your system requirements. The bold entries on the following screen illustrate how you would use the prompt line to make changes to two parameter lines.

The hardware screen chapter of the *iRMX® I Interactive Configuration Utility Reference Manual* explains how to respond to the specific prompts shown in this screen. The purpose of this section is to explain how to make entries on this and other types of screens.

```
(HARD)    Hardware

(BUS) System Bus Type [1 - MBI / 2 - MBII]              1
(CPU) Processor used in the System [0,1,2,3,4]          2
(OSP) 80130 Component used [Yes/No]                     NO
(TT)  Timer Type [1,2,3]                                1
(TPA) Timer Base Port Address [0-0FFFFH]                0D0H
(TPS) Timer Port Separation [0-0FFH]                    0
(TCN) Timer Counter Number [0,1,2]                      0
(CIL) Clock Interrupt Level [0-7]                       0
(CIN) Clock Interval [0-65535 msec]                     10
(CF)  Clock Frequency [0-65535 khz]                     1229
(GC)  Global Clock [546/CSM/86C38/None]                 NONE
(GCN) Global Clock Name [1-13 Chars]                    T546_CC
(NPX) Numeric Processor Extension [Yes/No]              YES
(NIL) 8087 NPX Interrupt Level [Encoded]                08H
(IF)  Initialize On-board Functions [1,2,3,4/No]        1
(RMB) Remote Boot Location [0=NONE, 40H-0FFFFH]         0H

Enter [Abbreviation = new value / Abbreviation ? / H ] :
:cil=4<CR>
:npx=no<CR>
:<CR>
```

```
(HARD)    Hardware

(BUS) System Bus Type [1 = MBI / 2 = MBII]           1
(CPU) Processor used in the System [0,1,2,3,4]       2
(OSP) 80130 Component used [Yes/No]                  NO
(TT)  Timer Type [1,2,3]                             1
(TPA) Timer Base Port Address [0-0FFFFH]             0D0H
(TPS) Timer Port Separation [0-0FFH]                 0
(TCN) Timer Counter Number [0,1,2]                   0
(CIL) Clock Interrupt Level [0-7]                    4
(CIN) Clock Interval [0-65535 msec]                  10
(CF)  Clock Frequency [0-65535 khz]                  1229
(GC)  Global Clock [546/CSM/86C38/None]              NONE
(GCN) Global Clock Name [1-13 Chars]                 T546_CC
(NPX) Numeric Processor Extension [Yes/No]           NO
(NIL) 8087 NPX Interrupt Level [Encoded]             08H
(IF)  Initialize On-board Functions [1,2,3,4/No]     1
(RMB) Remote Boot Location [0=NONE, 40H-0FFFFH]      0H

Enter [ Abbreviation = new value / Abbreviation ? / H ]
:
```

1.10.2 Entering File Names, Address Values, and Integer Constants

You can enter several types of values in response to a parameter line, depending on the range of values for the parameter. The kinds of values you can enter include

device/file name	A device or file name can be any device or file name acceptable to the operating system. The ICU converts the name to all uppercase characters. If you do not want the characters in a name converted to uppercase, enclose the name in single quotes.
integer constants	Constants must be unsigned integers that you can enter in any of three radices: decimal, hexadecimal or kilobyte. A trailing radix character indicates the radix of the number, as shown in Table 1-3. The default radix is decimal.
addresses	Address values must be entered in the form BASE:OFFSET. The radix must be specified (either explicitly or by default) for both portions of an address. For example, you must specify the selector of 900H and an offset address of 384H as 900H:384H.

Table 1-3. Integer Constant Formats

Radix	Trailing Character
Decimal	None or D
Hexadecimal	H or h
Kilobytes	K or k

1.10.3 Help Messages

The ICU provides three types of help messages to supply information and save you time as you are defining your definition files.

- For HELP about parameters, enter the parameter abbreviation followed by a "?".
- For HELP about the screen being displayed, enter a ?.
- For HELP about editing screens, enter ^H or H.

After reading the help messages, enter a carriage return to return to the screen you were editing.

1.10.4 Screen Formats

Three basic types of screen formats are used in the ICU: the fixed screen, the repetitive screen, and the repetitive-fixed screen. These screen formats have similar features.

1.10.4.1 Fixed Screen Formats

The fixed screen format enables you to make changes by entering the two- or three-letter abbreviation, the equal sign (=), the new value, and a carriage return. The "Hardware" screen shown earlier in this chapter is a fixed format screen.

1.10.4.2 Repetitive Screen Formats

Most screens use the fixed format to display information. However, a screen such as the "Prefixes" screen, shown below, uses a repetitive screen format. In a repetitive screen format, the same prompt is repeated many times. Each time you enter information on the screen, you define new system information. In the example below, each time you enter a line of information, you define a logical name for a directory. As you can see from the example, identifying numbers precede each line of information.

You can change this screen in the following ways:

- To delete a line, enter "^d" followed by the number of the line you want to delete. All lines that followed the deleted line are moved up one position in the list.

- To add a new line, enter the line number followed by an equal sign (=), the new value, and a carriage return. If the line number already exists, the information on the existing line and all lines following it will be moved down one position in the list.
- To replace the information in an existing line, delete the existing line, then add a new line using the same line number (as described above).

```
(PREF)      Prefixes

    Prefix = 1-45 characters
[1] Prefix = :PROG:
[2] Prefix = :UTILS:
[3] Prefix = :SYSTEM:
[4] Prefix = :LANG:
[5] Prefix = :ICU:
[6] Prefix = :$:
[7] Prefix =

Enter Changes [Number = new value / ^D Number / ? / H ]
:
```

1.10.4.3 Repetitive-Fixed Screen Formats

The repetitive-fixed screen format combines the features of the other two screen types. It repeats a full screen of information any number of times. In the following example, the "User Jobs" screen, you define a user job by entering information on the screen. When you complete this screen or any repetitive-fixed screen, a one-line query screen is displayed. In this case the query screen asks: "Do you have any/more User Jobs?". If you answer "yes" or "y", the ICU presents another "User Jobs" screen. Each time you make entries to one of these screens you define a new user job. The ICU repeats this screen until you respond with a "no", "n", and/or a carriage return to the prompt.

```
(USERJ)      User Jobs

(NAM) Job Name [0-14 Char]
(ODS) Object Directory Size [0-3840]                     0
(PMI) Pool Minimum [20H-0FFFFH]                          060H
(PMA) Pool Maximum [20H-0FFFFH]                          0FFFFH
(MOB) Maximum Objects [1-0FFFFH]                         0FFFFH
(MTK) Maximum Tasks [1-0FFFFH]                           0FFFFH
(MPR) Maximum Priority [0-255]                           129
(EHS) Exception Handler Address [CS:IP]                  0000:0000H
(EM)  Exception Mode [Never/Prog/Environ/All]            NEVER
(PV)  Parameter Validation [Yes/No]                      YES
(TP)  Task Priority [0-255]                              155
(TSA) Task Start Address [CS:IP]                         0000:0000H
(DSB) Data Segment Base [0-0FFFFH]                       0300H
(SSA) Stack Segment Address [SS:SP]                      0000:0000H
(SSI) Stack Size [0-0FFFFH]                              0300H
(NPX) Numeric Processor Extension Used [Yes/No]          NO

Enter [Abbreviation= new value / Abbreviation ? / H ]
: <CR>
```

1.11 SCREEN EDITING COMMANDS FOR THE ICU

Several special commands are available to simplify the editing process. They are summarized in Table 1-4 and then explained in detail in the following paragraphs. The commands are initiated by entering the caret "^" control character (or a character you substituted for the caret using the "Replace" command in command mode) followed by one or more characters. It is also possible to enter all of the commands, except Insert, Copy, and Delete, without the control character. If you try to use Insert or Delete without the control character, you will receive a message explaining the correct invocation of these commands. If you try to use Copy without the control character, you will receive an error message. Each command sequence must be terminated with a carriage return.

Command	Meaning	Screens Affected Fixed	 Repetitive	 Repetitive -fixed
^B or B	Back up to previous screen	X	X	X
^C or C	Return to command mode	X	X	X
^D	Delete a screen			X
^D <number>	Delete the element with this number		X	
^F <scabv> or F <scabv>	Find and display the specified screen	X	X	X
^H or H	Display the list of special commands that apply to the current screen format	X	X	X
^I	Insert an new screen in front of the current screen			X
^I <number>	Insert a new line			X
^CO	Copy the current screen			X
^R or R	Redisplay the current screen	X	X	X
^N or N	Go to the next logical screen	X	X	X
^S <string> or S <string>	Search the remaining screens for the specified string			X

Complete descriptions of the special editing commands are as follows:

^B or B	Enables you to move backwards from the current screen to the previous screen. The ICU displays the previous screen and enables you to continue as usual. Moving backwards beyond the beginning of the definition file returns you to command mode. This command can be used on all types of screens.
^C or C	Returns you to command mode from any ICU screen. It then displays the main menu.
^D	Enables you to delete an entire repetitive-fixed format screen. The screen deleted is the current screen.
^D <number>	Enables you to delete a specific item in a repetitive screen. The number you enter identifies the entry to be deleted.

^F <scabv> or F <scabv>	Finds and displays the screen indicated by the screen abbreviation. The syntax of the ^F command is ^F (or F) screen-abbreviation where the screen-abbreviation can be any abbreviation listed in Table 1-2. This command enables you to jump from one screen to another. If you specify a screen name not previously defined, this command jumps to the next available screen, and displays this warning message: `*** WARNING - The screen requested cannot be displayed` If you do not specify a screen abbreviation, the list of screen names and abbreviations is displayed (see Table 1-2) and you are prompted for a screen abbreviation. If you want to exit this command without entering an abbreviation, press the carriage return and continue to the next logical screen. Figure 1-4 shows a flowchart of how you proceed from one screen to the next if you simply enter a carriage return.
^H or H	Displays the list of special editing commands.
^I	Enables you to insert an additional repetitive-fixed screen in front of the current screen. Otherwise, the command ^I has no effect.
^I <number> = or <number> =	Enables you to add a new line to a repetitive screen. The ^I is optional. Only the line number and an equal sign are required.
^CO	Enables you to insert an identical copy of the current screen in front of the present screen. This command can be used only with a repetitive-fixed screen.
^R or R	Redisplays the current screen, showing any changes made. Entering ^R is the same as entering a null carriage return. The default or previously entered responses are displayed until you enter the ^R command (or <CR>) to show the changes you have made to this screen. If you are in a help screen, the command ^R returns you to the last non-help screen you were on.
^N or N	Displays the next logical screen. For example, if you are entering data on a unit-information screen and enter ^N, the first DUIB screen for that driver is displayed. If you enter ^N again, the first screen of the next driver is displayed, and so on. If you enter ^N in the last screen, the ICU returns to command mode.

^S <string> or S <string>	Searches repetitive-fixed screens of the same logical type for the specified string. When this command is entered, the search begins in the next screen of that logical type and searches all fields with a character range (for example, 1-31 characters). The search continues until a match is found. If no match is found, the cursor remains at its current position and the ICU displays the following message:

```
No next match found
```

The syntax for this command is

^S (or S) <string>

The following example shows how to use the ^S command. Assume you have 20 DUIB screens for the Mass Storage Controller driver and you want to find the screen that defines the device name as w0. First, you would get to the first "(I214)" screen. Then you would enter

```
^S w0 <CR>
```

The ICU searches all the DUIB Mass Storage Controller screens. If it finds a screen with "w0", it displays that screen. If it does not find "w0", it displays the "No next match found" message.

1.11.1 Deleting Data on a Repetitive Screen Format

To delete information from a repetitive screen, you must use the ^D <number> command, where <number> is the number of the line to be deleted. After the line is deleted, the remaining lines are renumbered and the screen is displayed again. The ICU does not allow you to delete a line that is not displayed. To replace a line you must first delete the existing line, and then insert the new line.

An example of how to delete data on a repetitive screen follows. Assume the "Prefix" screen is defined as shown below. The cursor is positioned under the word "Enter". If you wish to delete line 7, you would do so as shown here.

```
(PREF)     Prefixes

    Prefix = 1-45 characters
[1] Prefix = :PROG:
[2] Prefix = :UTILS:
[3] Prefix = :SYSTEM:
[4] Prefix = :LANG:
[5] Prefix = :ICU:
[6] Prefix = :$:
[7] Prefix = :WORK1:
[8] Prefix = :TMP86:
[9] Prefix =

Enter Changes [Number = new value / ^D Number / ? / H ]
:^d 7 <CR>
```

After line 7 is deleted, the screen is redisplayed with lines 8 and 9 renumbered to 7 and 8 as shown here.

```
(PREF)     Prefixes

    Prefix = 1- 45 characters
[1] Prefix = :PROG:
[2] Prefix = :UTILS:
[3] Prefix = :SYSTEM:
[4] Prefix = :LANG:
[5] Prefix = :ICU:
[6] Prefix = :$:
[7] Prefix = :TMP86:
[8] Prefix =

Enter Changes [Number = new value / ^D Number / ? / H ]
```

1.11.2 Inserting Data on a Repetitive Screen Format

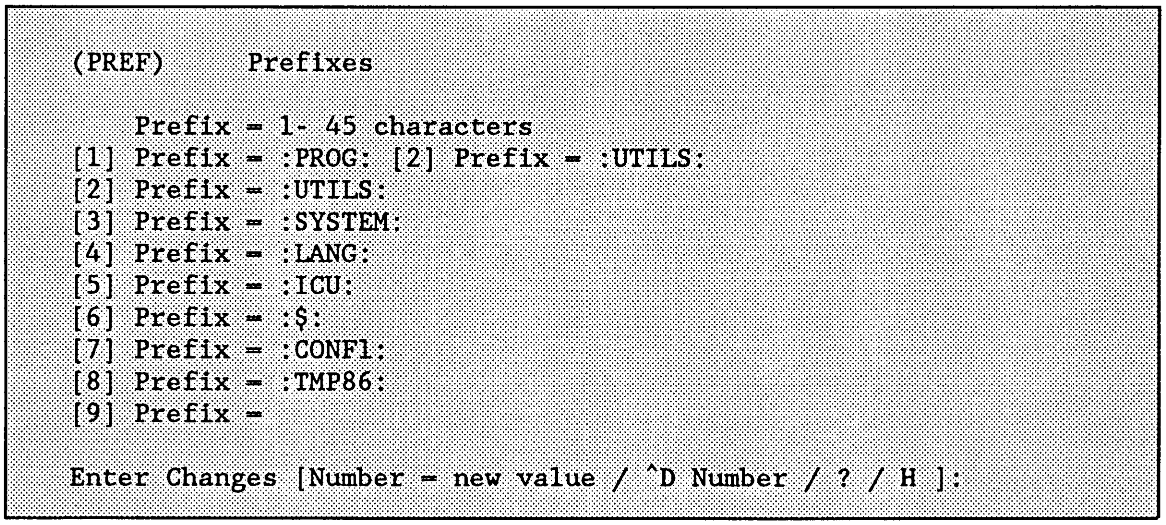
To insert a line on a repetitive screen, enter the insert command ^I (optional), the line number, an equal sign (=), and the new value. When the new line number is inserted, the ICU renumbers the remaining lines and displays the screen again. If the number you enter is larger than the actual number of lines in the screen, the ICU inserts the new line as the last line. Assume you want to insert a new prefix on line 7 of the "Prefix" screen displayed previously. You can enter

```
7 = :confl: <CR>
```

or

```
^I 7 = :confl: <CR>
```

and the screen will be redisplayed with the new values as shown here.



```
(PREF)      Prefixes

    Prefix = 1- 45 characters
[1] Prefix = :PROG: [2] Prefix = :UTILS:
[2] Prefix = :UTILS:
[3] Prefix = :SYSTEM:
[4] Prefix = :LANG:
[5] Prefix = :ICU:
[6] Prefix = :$:
[7] Prefix = :CONF1:
[8] Prefix = :TMP86:
[9] Prefix =

Enter Changes [Number = new value / ^D Number / ? / H ]:
```

If you are entering numerical data on a repetitive screen such as the "RAM Memory" screen, you can enter the data in any order. However, the ICU automatically arranges your data in the proper order and displays it on the screen. For example, if you enter the following three insert commands

```
^I 1 = 2000H, 4000H  <CR>
^I 2 = 80000H, 90000H  <CR>
^I 3 = 10000H, 12000H  <CR>
```

on the "RAM Memory" screen (see the *iRMX® I Interactive Configuration Utility Reference Manual*), the ICU sorts the data in ascending order and redisplays the lines as follows:

```
1 - 2000H, 4000H
2 - 10000H, 12000H
3 - 80000H, 90000H
```

1.11.3 Deleting a Repetitive-Fixed Screen

The ^D command enables you to delete information for an entire repetitive-fixed screen; you delete the current screen. You can use this command to delete I/O Jobs, User Jobs, OS Extensions, and Remote File Servers, as well as Intel and user devices. If you want to delete a device driver, it is only necessary to delete the Driver screen for that device. The ICU automatically deletes all the Unit and DUIB screens associated with it (see the *iRMX® I Interactive Configuration Utility Reference Manual* for more information).

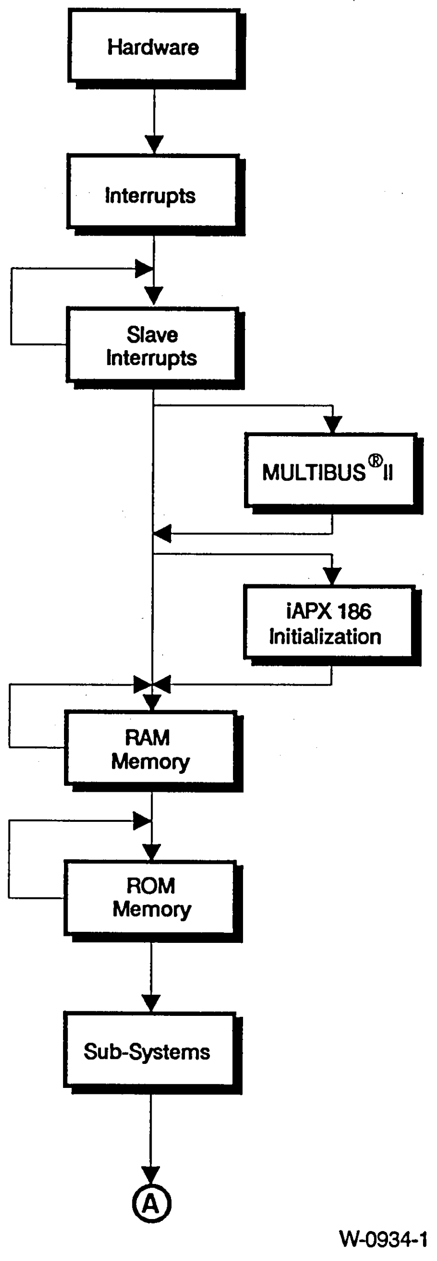


Figure 1-4. ICU Flowchart

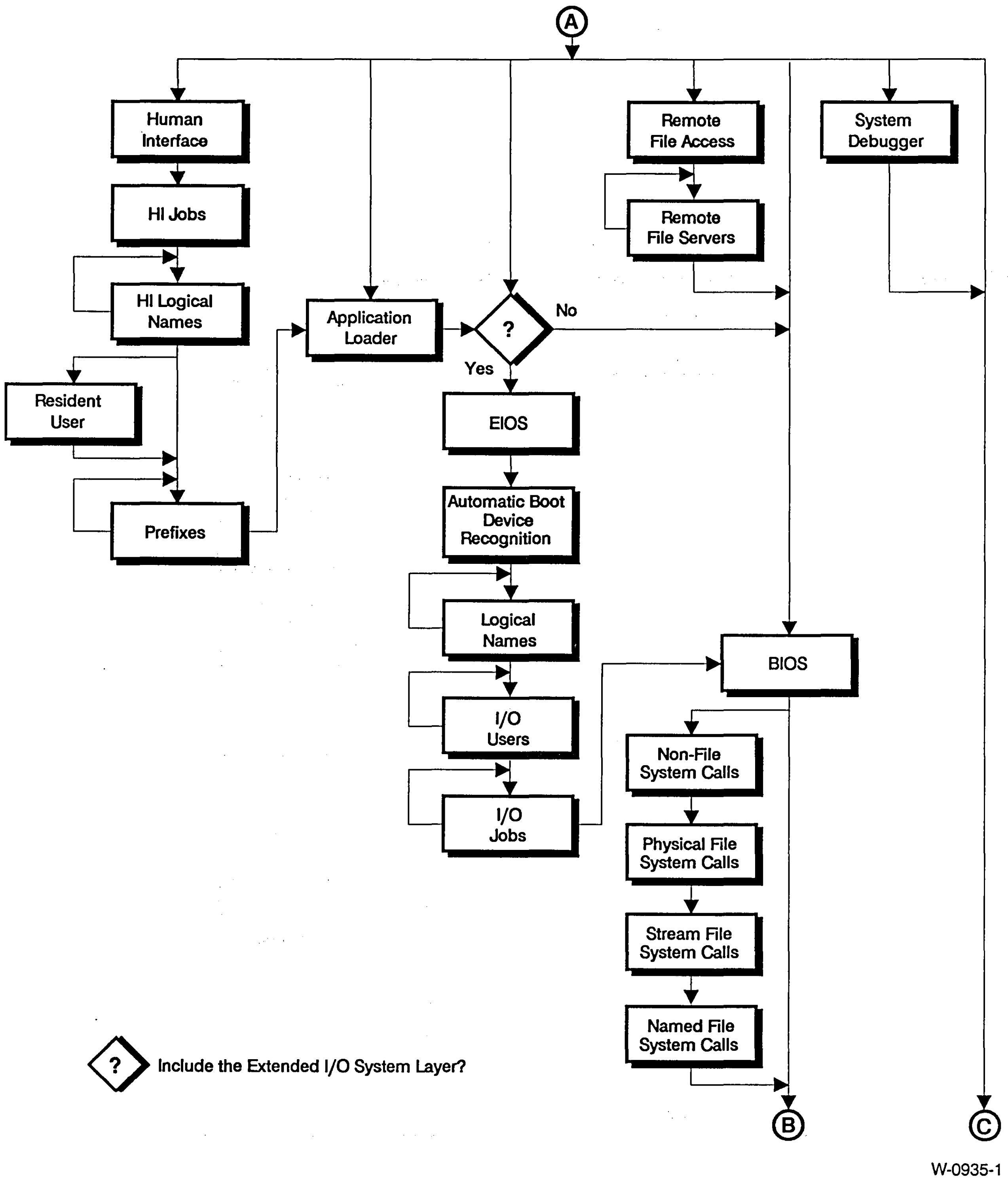


Figure 1-4. ICU Flowchart (Continued)

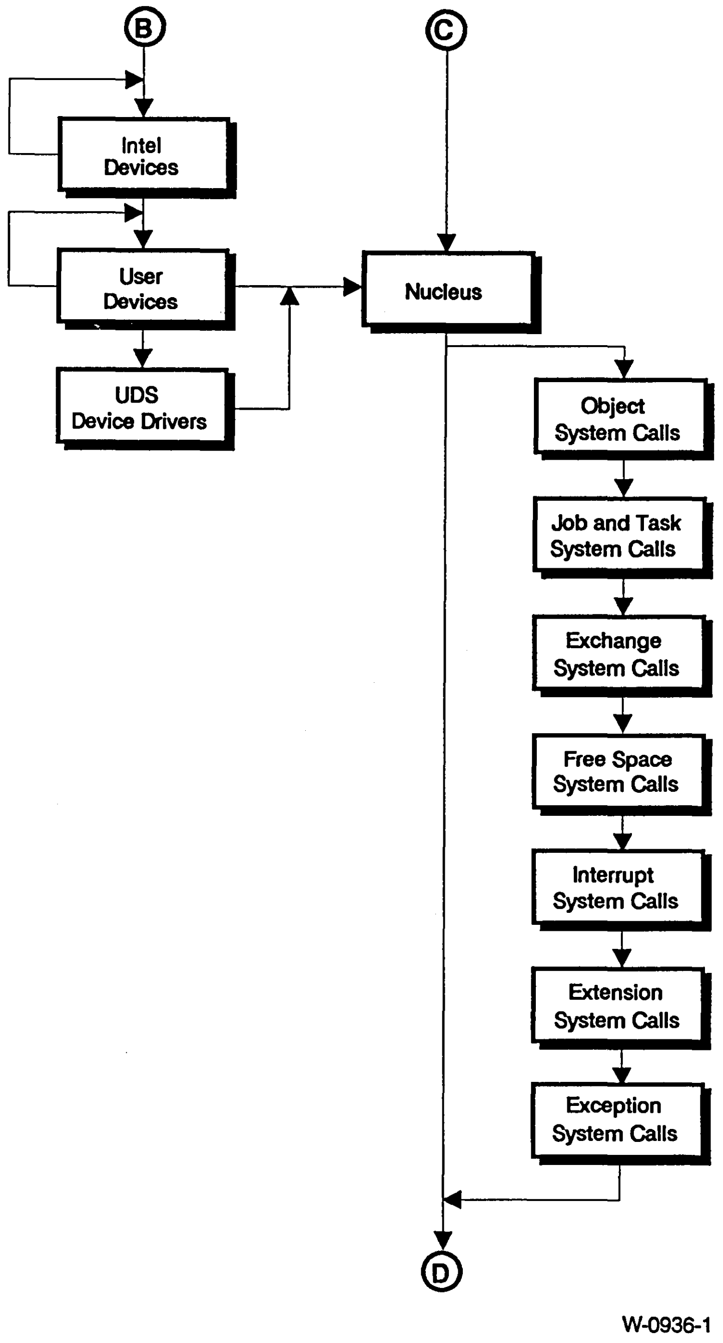


Figure 1-4. ICU Flowchart (Continued)

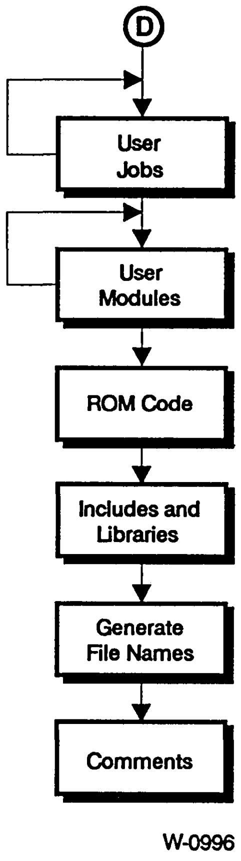


Figure 1-4. ICU Flowchart (Continued)

1.11.4 Inserting a Repetitive-Fixed Screen

The ^I command enables you to insert an additional screen of information between two existing screens. (This can be used only with the repetitive-fixed screen format.) Use this command on the screen you wish to precede. For example, if you have three User Jobs and wish to insert a fourth job between the second and third job, use the ^I command on the screen for the third job.

The copy command (^CO) can also be used to insert an additional repetitive-fixed screen. The copy command inserts a copy of the current screen in front of itself. The only difference between the insert command and the copy command is that the copy command uses the current screen values rather than the default values.

1.12 ICU ERROR MESSAGES

During the interactive portion of the ICU process, two types of error messages can occur:

- interactive error messages
- internal ICU errors

The interactive messages are the most frequently encountered, and are self-explanatory. The ICU internal error messages should not occur. The following sections explain these errors in more detail.

1.12.1 Interactive Error Messages

The ICU accepts data that you enter only if it lies within the range of acceptable values. Usually, the range of acceptable values for a given prompt appears in brackets "[]" on the prompt line. If you specify a value outside the range of acceptable values, the ICU displays one of the following messages, depending on the kind of value it requires (all of these messages are preceded by *** ERROR -):

- number expected or number too large
- number is not within its range
- address expected
- the selector is not within its range
- offset in address is not a number
- string too long
- a prefix of a legal string expected
- 'Yes or No' expected
- the field is 'Req'; cannot be changed

- erroneous delimiter
- the line entered overlaps

When an error occurs, the ICU does not change the current value of the parameter. If the values you specify lie within the range of acceptable values, the ICU accepts them without checking their reasonableness. Therefore, if you enter values that cause the ICU to generate a nonfunctional version of the operating system, neither the interactive phase nor the generation phase of the ICU will flag these values as errors.

When the ICU leaves the change phase and returns to the initial menu screen, it performs a number of logical tests, such as checking that the reserved memory areas do not overlap. If it detects a logical error, the ICU issues a self-explanatory error message. You must then make the necessary corrections to your definition file or you will not be able to generate a working system.

1.12.2 Internal ICU Errors

If during execution the ICU encounters an internal error such as the Screen Master File or the Template file being corrupted, it displays the following message:

```
*** ICU Internal Error - <number[,s]>
```

where <number[,s]> can be either one number or two numbers separated by a comma. The numbers represent an internal code for the ICU and are not meaningful for the user. Internal ICU errors rarely occur, but if you should receive this error message, follow these guidelines.

1. First, assume your definition file has become corrupted, and try running the ICU again with a new definition file.
2. If Step 1 is not the solution, try running the ICU with a new Screen Master File and a new Template file. Default versions of these files are kept in the directory :CONFIG:default.
3. If neither of the above solve the problem, contact your local Intel sales office.

1.13 UPGRADING DEFINITION FILES

There are three reasons you may have to upgrade definition files.

- To make iRMX 86 Release 7.0 definition files compatible with iRMX I.8
- To add Intel-supplied changes
- To add user device drivers

To upgrade definition files created by the iRMX 86 Release 7.0 version of the ICU, use the UPDEF86 Utility.

To upgrade your iRMX I.8 definition files to include Intel-supplied changes or user devices, invoke the ICU with the definition file you want upgraded as input. iRMX I.8 definition files can have two formats:

- ICU standard format with a specific version number
- Backup format (ASCII) used by different versions

The ICU checks the version numbers (see section, "Invocation Error Messages", earlier in this chapter) and decides how to proceed. If it is possible to upgrade the definition file without restoring the backup information, the ICU prompts

```
Do you want to update the file? y/[n]
```

A response of "Yes" causes the ICU to upgrade the file as you input it. You can then proceed with the ICU as usual.

If the ICU must restore to upgrade the definition file (for example, if the ICU is invoked with a backup file or a definition file whose Intel version number differs from the ICU version number), it invokes the restore process and prompts you as follows:

```
Do you want to restore from the file? y/[n]
```

A response of "No" causes the ICU to stop executing. A "Yes" response means the ICU should restore the backup information contained in the file, and create a new version of the definition file.

If you enter "Yes" and the input file is the same as the output file, you are prompted

```
Enter new output file name:
```

If the output file exists, the ICU displays this message:

```
    File <output_file> exists.  OVERWRITE?  y/[n]:
```

While restore is operating, the ICU displays a series of asterisks (*) on the screen. If the restore operation reaches completion with no loss of data, the ICU displays the main menu and you proceed as usual. However, if an error is encountered, the ICU displays the following message and exits.

```
    *** ERROR while restoring
    The Definition File has been restored to file:  <file-name>.def
    Inspect the log file: <file-name>.log
```

The ICU writes the backup information that was not restored to a log-file. The log-file lists each screen name followed by any errors that occurred while restoring that screen. It also lists abbreviations of fields which were not restored. The log-file has the same name as the output file but with a ".log" extension. The log-file makes it easy to compare the backup definition file and the restored file to see which values were not restored. You should then run the ICU on the restored definition file and correct the fields in error. After that you can proceed as usual.

Assume that while restoring from file up0.def, the ICU was not able to restore the "CF" parameter on the "Hardware" screen. The log file would look like this:

```
ICU86 <version number> Restoring from file : up0.def  <date>  <time>

----        Screen : HARD       ----
*** ERROR - number expected
                                    In field :  CF

----        Screen : INT        ----
```

The error messages in the log-file are the same as the ICU interactive error messages.

This example shows only a portion of the log-file. However, the actual file lists all the screen names. The version number, date, and time in the heading are variables.

1.14 THE ICUMRG86 UTILITY

The ICUMRG86 Utility supplied with the ICU provides the ability to include configuration support for new drivers. The ICUMRG86 Utility allows you to

- Integrate new Intel device drivers with a previous version of the operating system
- Integrate user-written device drivers into the operating system

The ICUMRG86 Utility combines the main Screen Master File (ICU86.SCM) and the main Template File for System Generation (ICU86.TPL) with the Screen Master File (SCM) and Template Files (TPL) for the new driver.

If you are adding an Intel-supplied driver, both the SCM and TPL files are supplied with the update package. In addition to adding your device driver, the ICUMRG86 Utility updates the "Intel Device" screen to include the new device, and changes the help message that lists all the screen names. Upon completion, ICUMRG86 updates the Update version number.

If you are adding a user-written device, the SCM and TPL files were previously generated by the UDS86 Utility (see the *iRMX® Device Drivers User's Guide* for more information). Upon completion, ICUMRG86 updates the User version number.

After running ICUMRG86, the version numbers of the new ICU and your definition files are different. To continue using your definition files, invoke the ICU as usual. The ICU will check for version number consistency, and if necessary issue a warning and a prompt (see section "Invocation Error Messages", earlier in this chapter) to which you should respond "Yes". The ICU then updates your definition files and continues executing. Figures 1-5 and 1-6 give the logical flow of the ICUMRG86 Utility when adding either an Intel device driver or a user device.

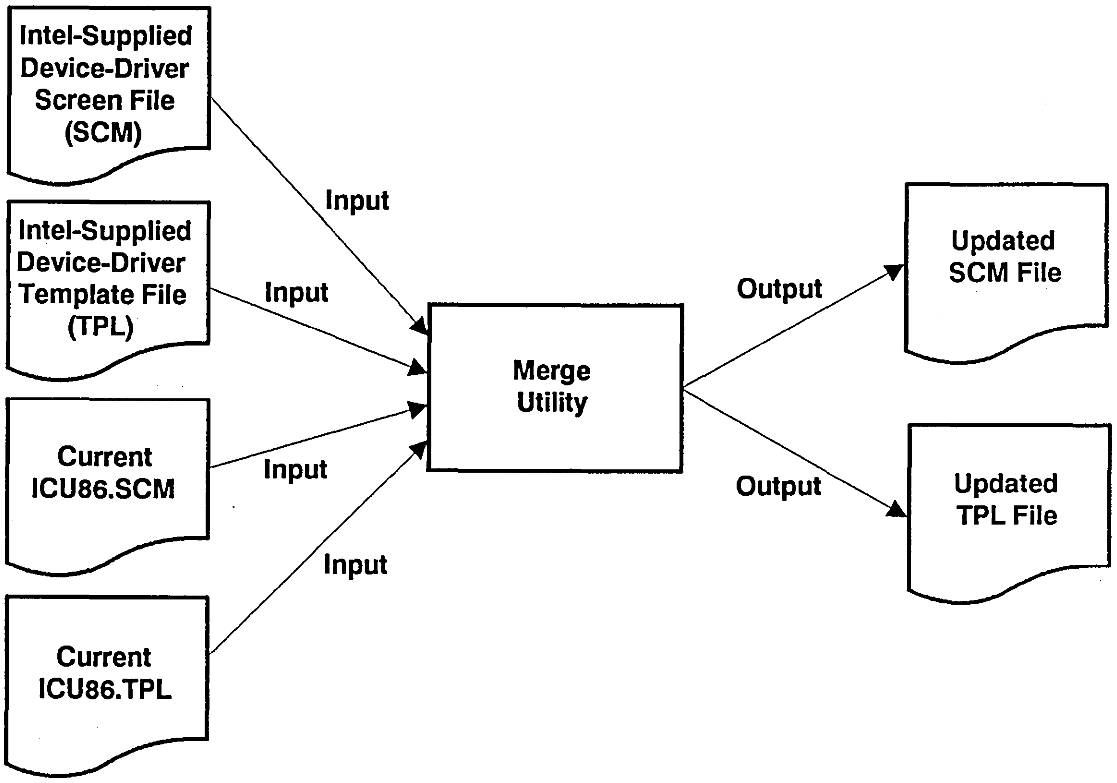


Figure 1-5. Merging Intel Device Drivers

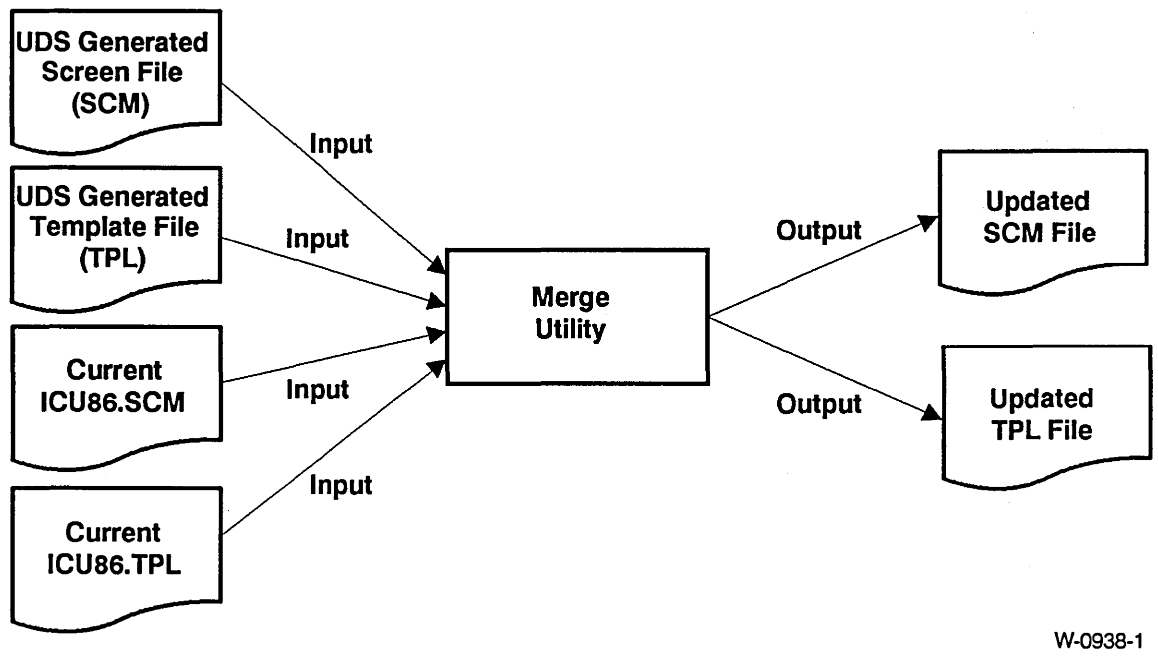


Figure 1-6. Merging User Devices

1.14.1 Invoking ICUMRG86

Before invoking ICUMRG86 be sure that the ICU86.SCM, ICU86.TPL, and ICUMRG86 files are in the same directory. To invoke the ICUMRG86 Utility enter

```
ICUMRG86  input-file(root) TO newicu-file(root)
```

where

input-file(root)	The input-file name without the extension providing the input to ICUMRG86. All extensions included in the pathname are ignored, and replaced by SCM and TPL. The ICUMRG86 Utility searches the input directory for input-file.SCM - contains all information about the new driver and the new "Intel Device" screen. input-file.TPL - contains information needed for generation of new screens. The ICUMRG86 Utility also uses ICU86.SCM and ICU86.TPL as input.
newicu-file(root)	The name of the two updated files, without their extensions, created by the ICUMRG86 Utility. All extensions included in the pathname are ignored, and replaced by SCM and TPL. ICUMRG86 creates newicu-file.SCM - contains the ICU Screen Master File updated with the new device driver. newicu-file.TPL - contains the ICU Template File updated with the new device driver.

Be aware that the ICUMRG86 utility always merges your .SCM and .TPL files with the ICU files ICU86.SCM and ICU86.TPL. If you plan to add support for several drivers to the ICU, make sure that the ICU86.SCM and ICU86.TPL files contain the latest version of your merged ICU files. Otherwise, ICUMRG86 will merge your driver information with outdated ICU files.

NOTE

Before changing the name of any ICUMRG86 output files to ICU86.SCM and ICU86.TPL, save the original files by copying them to other files (such as ICU86OLD.SCM and ICU86OLD.TPL). Although ICUMRG86 allows you to add support for new drivers, once you add that support, there is no way to remove it. If the device driver you added contains an error, you must revert back to the original .SCM and .TPL files.

1.14.2 ICUMRG86 Example

The following example shows how to add a device - D219.

```
ICUMRG86 D219 TO icunew <CR>
```

The input files are:	ICU86.SCM AND ICU86.TPL (located in same directory as ICUMRG86)
	D219.SCM and D219.TPL
The output files are:	ICUNEW.SCM and ICUNEW.TPL

Upon completion the system prompt is displayed. You are then ready to run the ICU and generate your system.

1.14.3 ICUMRG86 Error Messages

The ICUMRG86 utility generates an error message if one of the following occurs:

- it is not invoked correctly
- an I/O error occurs
- the version numbers are inconsistent
- either the SCM or TPL files are not valid

Invalid invocation of ICUMRG86 causes one of the following self-explanatory error messages to be displayed.

-
  ```
  parameters required
  USAGE:  ICUMRG86 infile TO outfile
  ```

-
  ```
  missing "TO outfile"
  USAGE:  ICUMRG86 infile TO outfile
  ```

-
  ```
  missing "TO"
  USAGE:  ICUMRG86 infile TO outfile
  ```

-
  ```
  missing "outfile"
  USAGE:  ICUMRG86 infile TO outfile
  ```

-
  ```
  too many parameters
  USAGE:  ICUMRG86 infile TO outfile
  ```

In addition to the invocation error messages, ICUMRG86 issues the error messages listed below.

-
  ```
  *** UDI Error - <exception-code>, <mnemonic>
  ```

  An error was detected by the UDI. The mnemonic explains the cause of the error. For example, you can receive this error message if ICUMRG86 cannot successfully change the extension.

-
  ```
  *** Error - input file same as output file
  ```

  The input and output files cannot be the same.

-
  ```
  *** I/O Error in file:  <file name>
     <excep-code>, <mnemonic>
  ```

  An I/O error occurred. For example, the ICUMRG86 utility was not able to create, open, read, write or seek one of the specified files.

-
  ```
  *** Error - <file name> is not a valid SCM file
  ```

  The data in the SCM file is not valid.

-
  ```
  *** Error - <file name> is not a valid TPL file
  ```

  The data in the TPL file is not valid.

-
  ```
  *** Error - inconsistency in the version of the internal ICU files
      Versions:                     INTEL     UPDATE   USER

                ICU86.SCM           <Intel> + <Update> <User Version>
                ICU86.TPL           <Intel> + <Update> <User Version>
  ```

  There is an inconsistency in the version numbers of the ICU86 SCM and ICU86 TPL files.

-
  ```
  *** Error - inconsistency in the version of the internal ICU files
      Versions:                     INTEL     UPDATE   USER

                Input Scm File      <Intel> + <Update> <User Version>
                Input Tpl File      <Intel> + <Update> <User Version>
  ```

  There is an inconsistency in the version numbers of the input SCM and TPL files.

- `*** Error - <screen-abbr> screen already exists in ICU86.SCM`

  Duplicate screen names are not allowed. You are probably merging the wrong SCM and TPL files, thus causing a duplicate name to be created.

- `*** Error - unexpected end of TPL file <file name>`

  An unexpected end of file in the TPL file was encountered.

GENERATING YOUR SYSTEM 2

2.1 INTRODUCTION

The process of generating your configured system consists of the following steps:

- Generating configuration files.
- Executing a SUBMIT file that compiles, assembles, links, and locates all necessary files.

2.2 GENERATING CONFIGURATION FILES

By using the ICU, you can define the operating system that best meets your individual needs. This process takes place while you are editing your definition file. When you have completely defined your system, return to command mode to generate your configured system as follows:

1. Use the List command to create a file that records your system configuration.
2. Use the Generate command to generate your configuration files.
3. Use Exit to save your changes and exit the ICU.

The following screen shows the results of having used the G[enerate] command to generate all the required configuration files (assuming the definition file used is newfile.def).

```
ENTER COMMAND : g

 ENTER a letter to be used as prefix :a

 The prefix letter is : A
 Beginning Nucleus File Generation
 ...................................................DONE
 Beginning Terminal Handler File Generation
 ...................................................DONE
 Beginning Basic I/O System File Generation
 ...................................................DONE
 Beginning Extended I/O System File Generation
 ...................................................DONE
 Beginning Application Loader File Generation
 ...................................................DONE
 Beginning Human Interface File Generation
 ...................................................DONE
 Beginning UDI File Generation
 ...................................................DONE
 Beginning System Debugger File Generation
 ...................................................DONE
 Beginning Submit File Generation
 ...................................................DONE

To create your application system, type:
    submit :$:NEWFILE.CSD
```

The files listed in Table 2-1 are the configuration files that define your system. The system processes these files during execution of your SUBMIT file. The ICU creates the SUBMIT file with the same filename as your definition file (with a .CSD extension). For example, the definition file used in the previous screen was labeled NEWFILE.DEF. Therefore, the SUBMIT file is called NEWFILE.CSD.

If you use the prefix option, be sure to choose a unique prefix each time you generate your system. If a file of the same name already exists, the ICU overwrites the old file with the new file.

Table 2-1 shows file names created using no prefix (carriage return only). If you enter any character other than carriage return when prompted for a prefix, that character is added as the prefix to the file names.

Table 2-1. Files Created by the G[enerate] Command

File Name	Screens Used to Define the File
NTABL.A86	Nucleus, Object Sys Calls, Job and Task Sys Calls, Exchange Sys Calls, Free Space Sys Calls, Interrupt Sys Calls, Extension Sys Calls, Exception Sys Calls
NDEVC.A86	Hardware, Interrupts, 80186 Initialization, MBII Hardware
MTHn.A86	Dynamic Debugger and Terminal Handler
ITABL.A86	BIOS, Non-File Sys Calls, Physical File Sys Calls, Stream File Sys Calls, Named File Sys Calls, Remote File Access
ICDEV.A86 and ITDEV.A86	All Intel and user devices, Remote File Access
ETABL.A86	EIOS Sys Calls
EDEVC.A86	EIOS, Automatic Boot Device, Logical Names
EJOBC.A86	I/O Users, I/O Jobs
HCONF.P86	Human Interface, HI Jobs, Resident User, Prefixes, HI Logical Names
LTABL.A86	None
LCONF.P86	Application Loader
SDBCN.A86	System Debugger
UTABL.A86	None
UDICN.A86	UDI
ROOT.A86	Hardware, Subsystems, MBII Hardware

CAUTION

Changes made to the ICU definition file are not reflected in your configuration files until you generate.

2.3 EXECUTING THE SUBMIT FILE

After you exit the ICU, execute the SUBMIT file and wait for your system to be generated.

The SUBMIT file assembles or compiles any configuration files generated by the ICU and links the object files with any needed libraries used by a subsystem. It then builds the system. The syntax for invoking the SUBMIT file is

```
SUBMIT output-file[.CSD] [TO filename] [echo]
```

where:

output-file	The name of your definition file.
filename	A file that the system creates to contain the output of the SUBMIT command.
e[cho]	Sends a copy of the data read to the screen.

For more information on the SUBMIT command, see the *Operator's Guide to the iRMX® Human Interface*.

2.3.1 Assembling the Configuration Files

The SUBMIT file generated by the ICU identifies the configuration files that must be assembled or compiled for each of your subsystems. The number of files assembled varies from system to system and depends upon the features that you choose. No errors should be encountered during this phase. Figure B-33, in Appendix B, gives an example of the SUBMIT file output during this phase of the configuration process.

2.3.2 Linking the Individual Subsystems

As soon as ASM86 generates the object files for a given subsystem, the SUBMIT file initiates LINK86 to link these object files together with any libraries needed by the subsystem. Any warnings generated during this phase should be ignored. Explanations of the various warnings appear at the end of this section. Figure B-33, in Appendix B, shows some of the output generated during this phase of the configuration process.

2.3.3 Warning Messages

When you invoke the system generation SUBMIT file generated by the first stage of the ICU, you might notice a number of warning messages generated by LINK86 or LOC86. Some of these messages are normal messages that you can ignore. This section lists such messages and the conditions under which they can be ignored.

- `WARNING 12:  UNRESOLVED SYMBOLS`

  This warning indicates that this link did not resolve all the external symbols declared. This situation is expected when configuring a system which includes the Basic I/O System (BIOS).

  The system generation SUBMIT file created by the ICU calls LINK86 several times when linking the BIOS. One of the last calls to LINK86 resolves the previously unresolved external references.

-
  ```
  WARNING 28:  POSSIBLE OVERLAP
    FILE:   <filename>
    MODULE:  <module name>
    SEGMENT:  <segment name>
    CLASS:  <class name>
  ```

  This warning indicates that the linker is combining two absolute segments. Again, this situation is expected.

- `WARNING 64:  PUBLIC SYMBOLS NOT SORTED DUE TO INSUFFICIENT MEMORY`

  This message indicates that your computer system does not have enough memory to allow the linker to sort the public symbols for the listing. If the linker returns this message, the public symbols are listed in the order that they occur. The iRMX I system generated will not be adversely affected if the listing is not alphabetically sorted. If you desire a sorted listing, you should increase the memory available to the user. The memory available to a particular user is defined in the :CONFIG:USER/username file. For details, refer to the *Operator's Guide to the iRMX® Human Interface*.

-
  ```
  WARNING 26:  DECREASING SIZE OF SEGMENT
    SEGMENT:  <segment name>
  ```

  This warning occurs because the system generation SUBMIT file specifies a SEGSIZE control on the LOC86 command line, and the size it specifies is smaller than the actual segment size. This warning is normal and should be expected. It usually occurs when the SUBMIT file contains a SEGSIZE(STACK(0)) control.

-
  ```
  WARNING 63:  SS AND SP REGISTERS NOT INITIALIZED
  WARNING 64:  DS REGISTER NOT INITIALIZED
  ```

  These warnings occur because the system generation SUBMIT file specifies an INITCODE control on the LOC86 command line, but the SUBMIT file does not initialize the SS, SP, and DS registers. This situation occurs when locating the root job, because the ICU correctly assigns the stack segment a length of zero. These warnings are normal.

-
  ```
  WARNING 10:  DIFFERENT VALUES FOR
    FILE:   :F1:NUC4.LIB
    MODULE: INVALID
    SYMBOL: VALIDATE_PARAMS
  ```

  This warning indicates that you have configured a system that does not use all the Nucleus system calls.

-
  ```
  WARNING 66:  START ADDRESS NOT SPECIFIED IN OUTPUT MODULE
  ```

  This warning indicates that there is no start address for the module being located. This warning message is expected when locating systems configured by the ICU. The SUBMIT file calls LOC86 several times to locate the different layers of your system. Only the root job's object file includes a start address.

2.3.4 Second-Stage of the ICU

After the SUBMIT file has directed the assembling and linking of each of the subsystems, the SUBMIT file initiates the second stage of the ICU. During this second stage, the ICU calculates where each module is to be located in memory blocks that you declared. The ICU does this by sorting the memory blocks by size, and searching from largest to smallest for the best fit for the largest module that it must locate. If any memory within the memory block chosen for this module still remains, the ICU resorts all of the remaining memory blocks by size (including the new block just left). The second and remaining modules are located by the ICU from largest to smallest in the same fashion.

The second stage of the ICU, ICU86.862, generates two additional submit files: ICULOC.CSD and ICUROT.CSD. The ICUROT.CSD submit file links and locates the Root Job.

2.3.5 Error Messages

In addition to warning messages, the language utilities and the second stage of the ICU (ICU86.862) can return error messages. Error messages are not normal and should not be ignored. They indicate a serious problem that will prevent the successful generation of your system. The following error messages can appear.

- `0021: E$FILE NOT EXIST`
  `8042: E$NOT CONNECTION, command aborted by EH`

  LIB86 can return one of these error messages when it specifies an invalid pathname for the file that it creates to contain your iRMX I library. One of these messages might also appear if you entered an invalid pathname as input to the ICU.

- `0021: E$FILE NOT EXIST`

  LIB86 can return one of these error messages when the files it tries to add to the iRMX I library do not exist. This situation might occur if you abort the system generation SUBMIT file and delete the files to be added to the library or if an error occurs during the assemble, link, or locate of the system modules. One of these messages might also appear if you enter an invalid pathname as input to the ICU.

The following error messages can be returned by the second stage of the ICU.

- `CANNOT CREATE FILE`

  The second stage of the ICU returns this message if it has difficulty creating one of files it must produce (ICULOC.CSD or ICUROT.CSD). This problem might occur under the Series IV operating system or iRMX I if these files already exist and have the write-protect attribute set, if there is no write, add or delete access to the directory, or if the disk is write-protected.

- `DESCRIPTION FILE IS NOT VALID`

  Either the definition file appearing on the command line is not a valid definition file, the file does not exist, or if the versions of the first and second stages aren't the same. This situation might occur if you accidentally delete the definition file (or copy over it) before invoking the system generation SUBMIT file.

- `MAP FILE NOT FOUND`

  The second stage of the ICU returns this message if it cannot find one of the .MP1 files created by LINK86. Under normal circumstances the .MP1 file should not be missing. However, the ICU might not find the .MP1 file under either of the following conditions:

  - LINK86 returned an error and did not create the .MP1 file.
  - The .MP1 file was deleted after it was created and before it was needed by the second'stage of the ICU. This situation can occur if you split the system generation SUBMIT file into two files and then delete the .MP1 file before invoking the second SUBMIT file.

- `BAD MAP FILE`

  Second stage of the ICU returns this error message if necessary information is not present in .MP1 file. This error may occur if the link of a subsystem fails after the map file is created but before the link completes.

- `NOT ENOUGH RAM FOR SYSTEM`

  The memory blocks you declared for RAM in the "RAM Memory" screen are not large enough for the system you defined.

- `NOT ENOUGH ROM FOR SYSTEM`

  The memory blocks you declared for PROM in the "ROM Memory" screen are not large enough for the system you defined.

2.3.6 Libraries for the System RAM and PROM Code

The SUBMIT file uses LIB86 to create two libraries. One library is for the system RAM code and the other library is for the system PROM code. As a final step, the SUBMIT file adds the subsystems generated to one of the two libraries.

PREPARING APPLICATION JOBS 3

3.1 INTRODUCTION

After you have prepared your application jobs and the subsystems, you should locate your first system entirely in RAM to enable you to test and debug of your programs. It is much easier to test and debug your programs in RAM than it is to reburn your PROMs when you detect errors. After debugging in RAM, you can locate the final system in PROM/RAM or copy it to a secondary storage device and load it with the Bootstrap Loader.

Putting together a RAM-based system consists of the following steps:

- Using the ICU to define your system
- Preparing your application code
- Planning where to locate your code
- Linking and locating the application jobs
- Using the ICU to generate the configuration files for the application system
- Loading and testing the system

These steps have not been numbered because conclusions that you draw while laying out your system affect where and when you locate your application jobs. This chapter describes the strategies that you need to consider when locating your system and explains how to use the ICU to generate your system. Both loading and testing your system are described in Chapter 5 of this manual.

3.2 PREPARING APPLICATION CODE

You can write the code for your application tasks in either PL/M-86 or assembly language. This manual assumes you are using PL/M-86. To use assembly language, you must adhere to the PL/M-86 calling conventions. These are described in the *ASM86 Macro Assembler* manual. The *iRMX® I Programming Techniques Reference Manual* also contains information to help you write assembly language tasks.

You must use Version 2.3 or later of PL/M-86. If you have any problems using the PL/M-86 language or compiling PL/M-86 code, refer to the *PL/M-86 User's Guide*. However, to use the features of the iRMX I Operating System, you must also follow the instructions in this chapter when writing your code.

3.2.1 Language Requirements

Adhere to the following language requirements when writing your task code:

- Make certain that any utilities you use are linked to the UDI libraries.
- In general, you should designate all of your tasks as procedures. Designation of initial tasks is the only exception to this recommendation. Refer to *iRMX® I Application Loader User's Guide* for details about main modules and procedures.
- If you are compiling your PL/M-86 code using any model other than LARGE, specify the ROM compiler control. This causes the compiler to place the CONST segment in the CODE class, where it can be more easily loaded into PROM. You do not need to specify the ROM control for those programs compiled using the LARGE model, because the compiler automatically does this for the LARGE model.
- Use the DATA and INITIAL statements with care. The DATA statement is valid only if you are using the PL/M-86 LARGE model of segmentation or if you specify the ROM compiler control. The INITIAL statement cannot be used in a procedure if you are going to place that procedure in PROM. It can be used, however, if you are going to use the Bootstrap Loader or the Application Loader to load the procedure into memory.

3.2.2 Include Files

A number of files must be present on your microcomputer system to compile your application software and to configure your operating system. The Includes and Libraries screen, discussed in the *iRMX® I Interactive Configuration Utility Reference Manual*, is used to select the files that must be present to configure your operating system. This section discusses the files that you need to compile your application software.

Any program containing iRMX I system calls must include an external declaration for each call used. Declarations for the system calls are provided for FORTRAN, Pascal, and PL/M-86 in files called INCLUDE files. When you install the system as described in the *iRMX® I Hardware and Software Installation Guide*, these files are located in the /RMX86/INC directory. FORTRAN system calls are in the file RMXFOR.EXT and Pascal system calls are in the file RMXPAS.EXT. Each PL/M-86 system call is declared in its own INCLUDE file and in an INCLUDE file for its subsystem of the operating system. The INCLUDE files for the subsystems are:

Subsystem	File Name
Nucleus	NUCLUS.EXT
BIOS	BIOS.EXT
EIOS	EIOS.EXT
Application Loader	LOADER.EXT
Human Interface	HI.EXT
UDI	UDI.EXT

If your PL/M application does not use all system calls in a particular subsystem, you may want to save memory by using the individual INCLUDE file for each system call, rather than using the INCLUDE file for the subsystem. To include the necessary files in the compilation of your procedures, use the PL/M-86 $INCLUDE control. The *PL/M-86 User's Guide* describes this control.

3.3 GENERAL SYSTEM LAYOUT

Your application system must include at least one application job or the Human Interface. If your application system includes an application job (a first-level job or an I/O job), you must decide where to locate the code for that job. When creating an initial system, you can locate your application code either high or low in RAM. This decision, as well as the decision to rearrange the location of your system code, is discussed in the sections that follow.

All systems, regardless of any other decision, have a minimum memory address at which they can start. Figure 3-1 shows the standard usage of memory from locations 0:0H to 119:FH for MULTIBUS® I systems. It is recommended that you adhere to this memory usage and start your system at 120:0H.

Notice in Figure 3-1 that addresses corresponding to the beginning and end of the interrupt vectors have been included. The locations 40:0H through FF:FH are reserved for the iSDM™ 86 or iSDM 286 monitors. Locations 100:0H through 119:FH are reserved for use by Intel disk controllers. Adhering to these recommendations for reserved addresses allows you to use the default addresses supported by the iRMX I Basic I/O System.

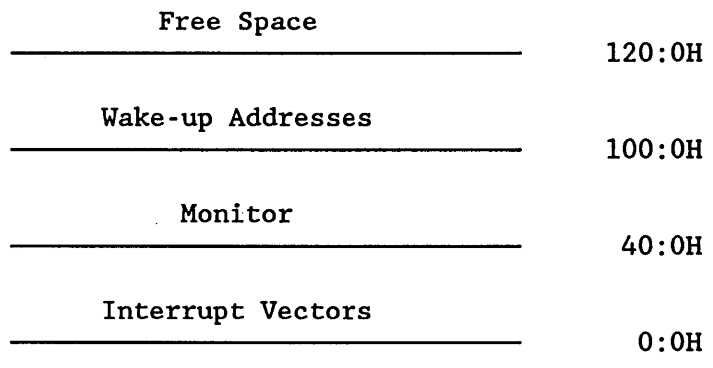


Figure 3-1. Reserved Memory Locations for MULTIBUS® I Systems

3.3.1 System Type

At first, when creating an initial test system, locate all of your modules in RAM. This allows you to lay out the system on a job-by-job basis. You can locate all segments associated with one job (code segments, data segments, etc.) sequentially in RAM and locate all segments of the next job following the first.

Later, if you locate a final PROM/RAM system, you must locate the system by class, not by job. You must locate the code classes from all of the jobs at PROM addresses, and the data, stack, and memory classes at RAM addresses. For details on locating a PROM/RAM system, refer to Appendix C and the description of the ICU's ROM Code screen in the *iRMX® I Interactive Configuration Utility Reference Manual.*

3.3.2 Low or High Location of Modules

You can locate your application software either low in memory (at 120:0H) or high in memory (at a numeric address greater than the operating system's highest address).

3.3.2.1 Locating Your Applications Low in RAM

When you locate the application software low in memory you must:

- Assign the first application job (module) to the numerically smallest memory location
- Use the ICU to declare the first available memory at a location greater than the highest memory used by your application code

Figure 3-2 illustrates locating your application software low in memory.

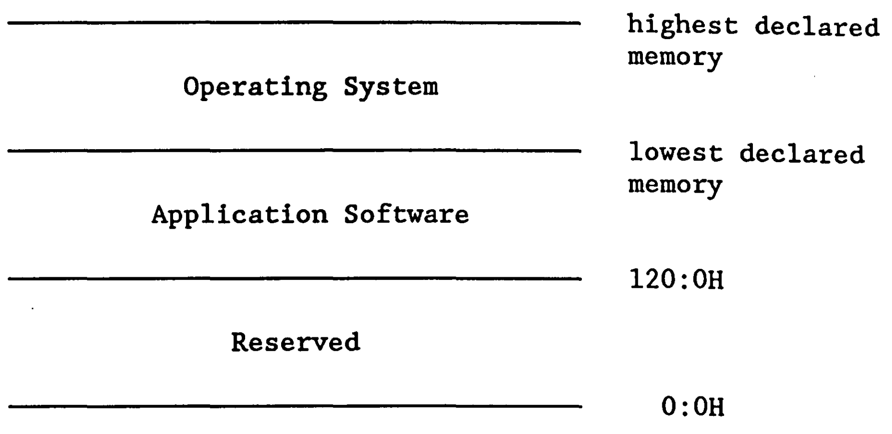


Figure 3-2. Locating Your Application Low in RAM

The major advantage in locating your application code low in memory is the ease in using the ICU. When you use this approach, you link and locate your application code and then run the ICU only once.

The disadvantages in locating your application code low in memory are as follows:

- First, even if you allow additional memory for growth of your application software, you may have to rerun the ICU frequently as you debug and augment your application software.
- Second, if your lowest address in RAM is on-board RAM, then you are not using your memory to your best advantage. (This may not concern you if your final system is PROM based.)

3.3.2.2 Locating Your Applications High in RAM

When you locate your application code high in memory, you must:

- use the ICU to declare 120:0H as the lowest memory for the operating system
- locate your application software at the first available address not declared for the operating system.

Figure 3-3 illustrates locating your application software high in memory.

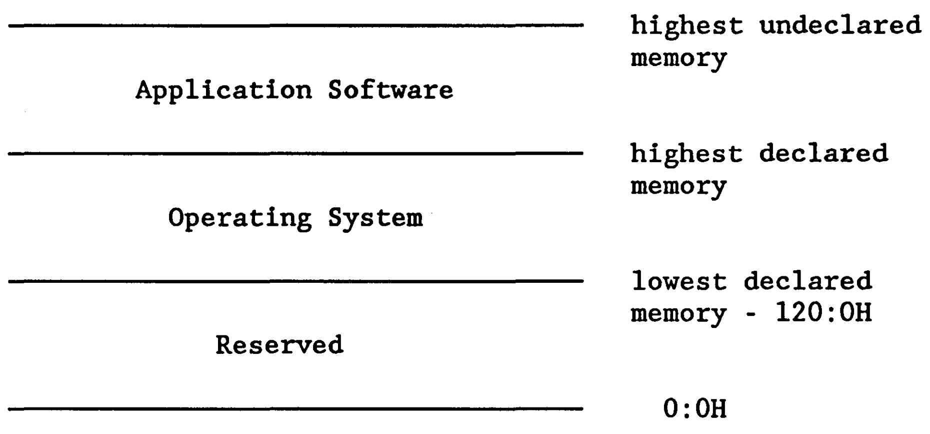


Figure 3-3. Locating Your Application High in RAM

The advantages in locating your application code high in memory are as follows:

- First, as your application software grows, the number of changes that require running the ICU is kept to a minimum. (You only have to change the addresses of your application jobs and not the declared memory addresses for the operating system.)
- Second, you can have the ICU locate part of your operating system in faster on-board RAM.

The disadvantages of locating your application code high in memory are as follows:

- You have to run the ICU twice. You run the ICU once before you locate your application code and a second time after you locate your application code. The first time you run the ICU you define and locate just your operating system. This allows you to determine the highest memory address used by your operating system. The second time you run the ICU to specify application code information and to redeclare your memory blocks.
- If you make large configuration changes to your system, you may have to move your application code.

3.3.3 Locating the Subsystems

As explained in Chapter 2, the ICU locates the system modules in order by size. This may not be the order in which you want your modules. In particular, if you want your Nucleus located at the lowest address in RAM, you must take additional steps to make this happen. The following steps summarize what you need to do.

1. Define your operating system using the ICU.
2. Locate your operating system using the ICU.
3. Use the locate maps generated in step 2 to ascertain the size of each of your system modules. Table 3-1 lists the file names for each system module.
4. Lay out a memory map of your proposed system. Allow 10 to 20 additional paragraphs (16 bytes each) per module for ICU second stage round-off errors.
5. Link and locate your application software.
6. Redefine your memory usage using the ICU. This step includes breaking your single block of RAM memory into a number of smaller blocks. You should declare one block for each system module. The size of the block should equal the size of the system module plus 20 paragraphs.
7. Specify the location of your application software using the ICU.
8. Assemble, link and locate your operating system using the ICU. This step is explained in detail later in this chapter.

Table 3-1. Locate Map File Names

Module	File Names
Nucleus	NUCLUS.MP2
Terminal Handler #1	MTH1.MP2
Terminal Handler #2 *	MTH2.MP2
Basic I/O System	IOS.MP2
Extended I/O System	EIOS.MP2
Application Loader	LOADER.MP2
Human Interface	HI.MP2
Root Job	ROOT.MP2
Universal Development Interface	UDI.MP2
System Debugger	SDB.MP2

Note: * Each Terminal Handler is assigned a number that corresponds to the order in which it was specified in the ICU. You can identify the Debugger by its relative Terminal Handler number.

3.3.4 Reading a Locate Map

One of the LOC86 options used while locating a system or an application program is the creation of a locate map for each module being located. These maps have filenames with the extension "MP2". Figure 3-4 shows a partial listing for ROOT.MP2. Note that the Root Job code starts at location 455C:0H and stops at location 45CB:FH. The Root Job data starts at location 4631:0H and stops at location 4632:3H. The Root Job stack starts at location 4632:4H and stops at location 4644:FH.

```
    iRMX I 8086 LOCATER, V2.5

    INPUT FILE: CROOT.LNK
    OUTPUT FILE: ROOT
    CONTROLS SPECIFIED IN INVOCATION COMMAND:
       TO ROOT SEGSIZE(STACK(0)) ORDER(CLASSES(DATA,STACK))
      PRINT(ROOT.MP2) ADDRESSES(CLASSES(CODE(0455C0H),DATA(046310H)))
      INITCODE(0455C0H) OC(NOLI,NOCM,NOSB) PC(NOLI,PL,NOCM,NOSB)
          •
          •
          •
    MEMORY MAP OF MODULE RBEGIN

    MODULE START ADDRESS  PARAGRAPH = 455CH  OFFSET = 0000H
    SEGMENT MAP

    START      STOP      LENGTH ALIGN NAME            CLASS        OVERLAY

--->455C0H   45BECH     062DH   W   CODE            CODE
    45BEEH   45BF9H     000CH   W   SAB_DESCRIPTOR  CODE
                                   -S
    45BFAH   45CBFH     00C6H   W   U_J_DESCRIPTOR  CODE
                                   -S
--->46310H   46323H     0014H   W   DATA            DATA
--->46324H   4644FH     012CH   W   INIT_STACK      STACK
    46450H   46450H     0000H   W   STACK           STACK
    46450H   46450H     0000H   G   ??SEG
    46450H   46450H     0000H   W   MEMORY          MEMORY


    GROUP MAP

    ADDRESS  GROUP OR SEGMENT NAME
    46310H   DGROUP
             DATA
    455C0H   CGROUP
             CODE
             SAB_DESCRIPTORS
             U_J_DESCRIPTORS
```

Figure 3-4. ROOT.MP2 File

3.3.5 Preparing a Memory Map Worksheet

Based on the information in a memory map we can prepare a memory map worksheet. These worksheets are used for planning memory use. Figure 3-5 shows a blank memory map worksheet.

iRMX® I RAM SYSTEM MEMORY MAP WORKSHEET

	:
	:
	:
	:
	:
	:
	:
	:
	:
	:
	0120:0H
iSBC 215 wake-up address	
	0100:0H
Monitor	
	0040:0H
Interrupt vector	
	0:0H

Figure 3-5. System Memory Map Worksheet

Figure 3-6 shows an example worksheet that has been filled in with the information from the ROOT.MP2 file. Note that since Chapter 2 points out that the Root Job is linked and located last, we can correctly assume that the Root Job has a higher address than the remaining system module code and data segments.

```
                 iRMX® I RAM SYSTEM MEMORY MAP WORKSHEET

______________________________________________________________  7FFF:FH
                              .
                              .
                              .
______________________________________________________________

                          Free Space
______________________________________________________________  ???????

                       Application Job
______________________________________________________________  4645:0H

                        Root Job Stack
______________________________________________________________  4632:4H

                        Root Job Data
______________________________________________________________  4631:0H

                        Root Job Code
______________________________________________________________  455C:0H

                Operating System Data and Code
______________________________________________________________  0120:0H

                   iSBC 215 wake-up address
______________________________________________________________  0100:0H

                           Monitor
______________________________________________________________  0040:0H

                       Interrupt vector
______________________________________________________________     0:0H
```

Figure 3-6. Sample Worksheet

3.4 LINKING AND LOCATING APPLICATION JOBS

Linking and locating your application jobs is an iterative process. That is, you must link and locate one application job and its offspring jobs, examine the locate maps to determine the ending address, and use that information to link and locate the next application job and its offspring.

The most common method of linking and locating your application jobs is to link the application job together with every job ultimately created by that application job and one or more interface libraries. You must then locate this module at an absolute address. Figure 3-7 illustrates this link and locate procedure.

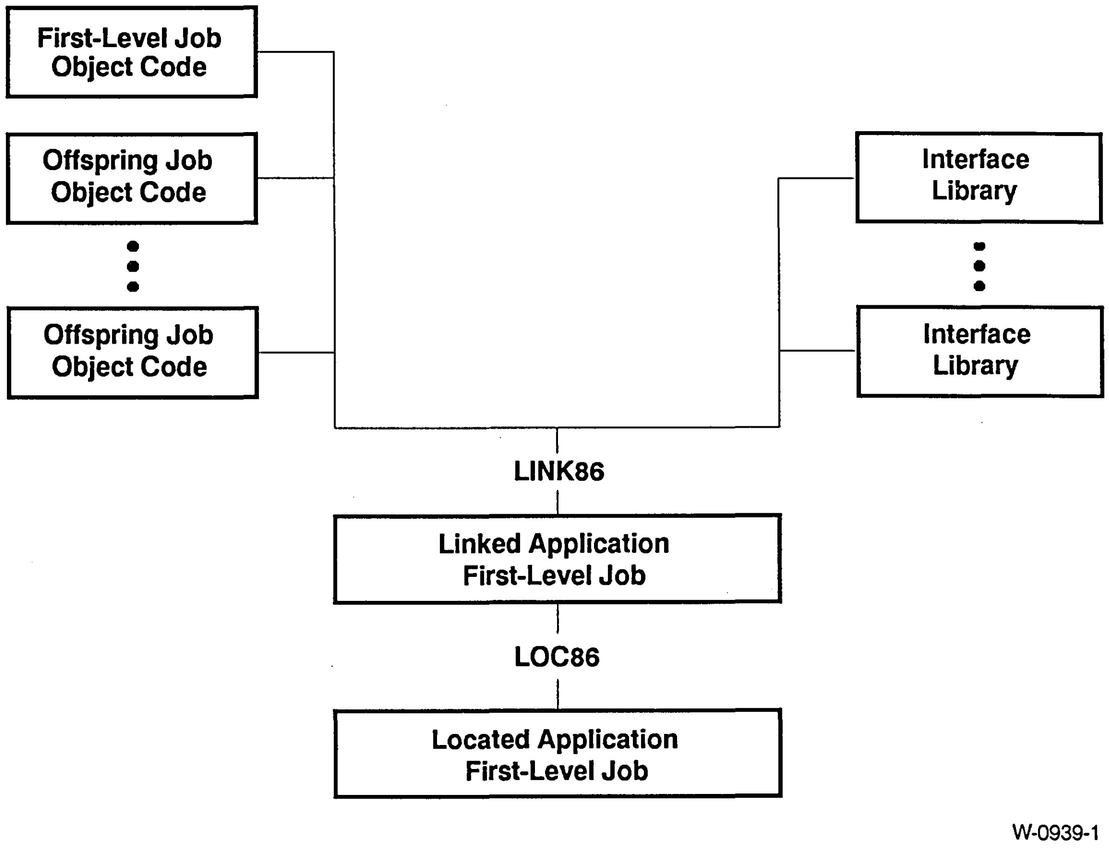

First-Level Job Object Code
Offspring Job Object Code
Offspring Job Object Code
Interface Library
Interface Library
LINK86
Linked Application First-Level Job
LOC86
Located Application First-Level Job
W-0939-1

Figure 3-7. Application Job Link and Locate Procedure

If you do not link your application jobs with their offspring jobs, you should link and locate the offspring jobs first. By doing this, you can obtain the absolute starting locations of the offspring tasks from the locate maps and specify these values in the CREATE$JOB (or CREATE$IO$JOB) and CREATE$TASK calls of their parent tasks before compiling the parents.

The following sections describe the individual link and locate commands in more detail, and describe the interface libraries.

3.4.1 Linking Application Jobs

The LINK86 command is used to link your application jobs. This command is described in detail in the *8086 Family Utilities User's Guide*. The example which follows shows the format of the LINK86 command as it is used in an iRMX I environment.

```
LINK86              &
     app job.obj,   &
     interface.lib  &
TO  app job.lnk  MAP  PRINT(app job.mpl)
```

where:

app_job.obj	Pathname of the file containing the object code for your application job. You do not need to provide this code in one file; you can link in several files or libraries at this point.
interface.lib	Pathname of the file containing the interface libraries for the subsystems that your jobs make use of. You may have to link in several libraries at this point. These interface libraries are described in later paragraphs of this section.
app_job.lnk	Pathname of the file in which LINK86 places the module containing your linked application code. Use this file as the input file when locating your application job.
app_job.mp1	Pathname of the file in which LINK86 writes the link map for the application job.

During the link process, you must link in a number of interface libraries. These libraries contain the routines that satisfy external references to system calls that you make in your application code. The number and names of the libraries that you must link in with your application code depend on which subsystems your jobs use and which model of segmentation the jobs were compiled under. Table 3-2 shows the correlation between subsystems, models of segmentation, and interface libraries. Specify these libraries as the last modules in the LINK86 input list so that they can satisfy references from all linked modules. Notice that no library exists for the SMALL model of PL/M-86 segmentation; except for Universal Development Interface (UDI) level applications, the iRMX I Operating System does not support applications compiled in SMALL.

Table 3-2. Interface Libraries as a Function of PL/M-86 Models and Subsystems

Subsystem	SMALL	COMPACT	LARGE or MEDIUM
Nucleus		RPIFC.LIB	RPIFL.LIB
Basic I/O System		IPIFC.LIB	IPIFL.LIB
Application Loader		LPIFC.LIB	LPIFL.LIB
Extended I/O System		EPIFC.LIB	EPIFL.LIB
Human Interface		HPIFC.LIB	HPIFL.LIB
UDI	SMALL.LIB	COMPAC.LIB	LARGE.LIB

3.4.2 Locating Application Jobs

After you have used LINK86 to generate a link module for your application job, you must use LOC86 to bind this link module to absolute addresses. The *8086 Family Utilities User's Guide* contains specific instructions on the use of the LOC86 command.

Since you are laying out your test system by job rather than by class, use a combination of the ORDER and ADDRESSES controls on LOC86 to simplify the location process. Use the ORDER control to declare the order in which the classes of the job are to be located. Then declare the absolute address of the code class with the ADDRESSES control. LOC86 automatically locates the rest of the classes following the code class. If you do this, a call to LOC86 appears similar to the following:

```
LOC86   input_file  TO  output_file                        &
        ORDER (CLASSES (CODE, DATA, STACK, MEMORY))        &
        SEGSIZE (STACK (stack_size))                       &
        ADDRESSES (CLASSES (CODE (absolute_address)))      &
        MAP  PRINT (map_file)                              &
        NOINITCODE                                         &
        OBJECTCONTROLS (NOLINES,NOCOMMENTS,NOPUBLICS,      &
                        NOSYMBOLS)
```

where:

input_file	Pathname of link file produced previously by LINK86.
output_file	Pathname of the file in which LOC86 writes the absolute module.
stack size	Size of this job's stack. Use this control for those jobs requiring a statically allocated stack. A stack is statically allocated if you, and not the operating system, specify a stack location and size. A minimum value of 200H should be specified, if this control is required; otherwise specify zero. It is recommended that you specify zero for this parameter and let the Nucleus dynamically allocate a stack whenever possible. This depends, however, on the model of PL/M-86 computation that you used when compiling your code. With dynamically allocated stacks, you specify the stack size in the ICU.
absolute_address	Absolute starting location of the code segment of the job. You can obtain this address by examining the locate map of the previously located module. You specify the starting location of the code segment for first-level jobs and I/O jobs in the ICU.
map_file	Pathname of the file in which LOC86 writes the locate map. Always generate the locate map. You need it in order to determine where to locate the next module.

Use this form of the LOC86 command to locate each application job.

3.5 LINKING AND LOCATING JOBS IN A RAM/PROM-BASED SYSTEM

In order to create a final RAM/PROM system, you should take the following steps:

- Minimize the memory address space requirements of your system by eliminating any padding factor you may have used when locating your jobs.
- Test your final system in RAM first, locate it into PROM/RAM, and burn the appropriate parts into PROM.

To shorten the time needed to load your complete system, burn your fully tested and completely debugged jobs into PROM while still testing and developing other jobs in RAM. Then, each time you reload your system, you need only load the jobs on which you are still working.

3.5.1 Minimizing the Memory Address Space

When you originally located your application jobs, you included padding factors in the calculations used to determine starting addresses of succeeding jobs. The additional space allocated with the padding factors allowed you to make small changes in your programs that increased their sizes without changing the LOC86 commands used to locate them. The modules, despite increasing in size, did not overlap each other. In your final system, you have already debugged all of your programs; their sizes are fixed, so now you can eliminate any extra space existing between modules, if you desire.

Follow the procedures outlined in a previous section "Locating Application Jobs" to locate your application's first-level and I/O jobs again. This time, however, leave out the padding factors between jobs. Then modify the ICU definition file by changing your responses as follows:

- Change the responses to prompts (TSA), (DSB), and (SSA) on the "User Jobs" screen and/or the "I/O Jobs" screen to reflect the new location of the job.
- Change the responses to the "RAM Memory" screen to reflect the smaller size of reserved memory.

After you have located your system, load it into RAM and test it again to make sure that it functions correctly.

You can perform this procedure in conjunction with the one described in the next section. However, it might be wise to perform them separately in order to localize any possible errors.

3.5.2 Locating the PROM/RAM-Based System

When you located your initial test and development system, you located it by job. That is, if you had three jobs, they were laid out as shown in Figure 3-8.

This was relatively easy; it allowed you to use the ORDER control in LOC86 and specify only one address for each job with the ADDRESSES control. However, when configuring any final system that will be programmed into PROM, you should lay out the system by class, not by job. All of the PROM-resident segments from all of the jobs should be positioned together. Likewise, all of the RAM-resident segments should be positioned together. Thus, if you had the same three jobs and were laying out a PROM/RAM system, you should structure your memory as shown in Figure 3-9.

```
____________________________________  high memory

         Job 3 MEMORY class
         Job 3 STACK class
         Job 3 DATA class
         Job 3 CODE class
____________________________________

         Job 2 MEMORY class
         Job 2 STACK class
         Job 2 DATA class
         Job 2 CODE class
____________________________________

         Job 1 MEMORY class
         Job 1 STACK class
         Job 1 DATA class
         Job 1 CODE class
____________________________________  low memory
```

Figure 3-8. Memory Layout of a RAM-based System

All of the code classes are located in the upper memory, or PROM, and the remainder are located in RAM.

As you can see, in order to transform your RAM-based system into a PROM/RAM system, you must locate your jobs again. Before you do that, however, you should prepare a memory map.

3.5.2.1 Preparing a Memory Map

To prepare a memory map, follow the procedures outlined in the "Locating the Subsystems," "Reading a Locate Map," and "Preparing a Memory Map Worksheet" sections of this chapter, with one exception. In this map, record not only the first available RAM address and the last available RAM address, but also the first available PROM address and the last available PROM address. You need this information on your memory map because for PROM/RAM systems, you must specify a location for both the PROM-resident code classes and the RAM-resident classes.

```
___________________________________  high memory

         Job 3 CODE class
         Job 2 CODE class               PROM
         Job 1 CODE class
___________________________________

         Job 3 MEMORY class
         Job 3 STACK class
         Job 3 DATA class
         Job 2 MEMORY class
         Job 2 STACK class              RAM
         Job 2 DATA class
         Job 1 MEMORY class
         Job 1 STACK class
         Job 1 DATA class
___________________________________  low memory
```

Figure 3-9. Memory Layout of a PROM/RAM System

3.5.3 Locating the Application Modules

The procedure for locating the application modules of a PROM/RAM system, like that for a RAM-based system, is an iterative procedure. You locate one module, record its addresses in the memory map, and use those values to determine where to locate the next module. The format of the LOC86 command used to locate these modules is slightly different from the one used to locate the RAM-resident system.

```
LOC86   input_file TO output_file                             &
        ORDER (CLASSES (DATA, STACK, MEMORY))                 &
        SEGSIZE (STACK (stack_size))                          &
        ADDRESSES (CLASSES (CODE (rom address),               &
            DATA (ram_address)))                              &
        MAP PRINT (map_file)                                  &
        NOINITCODE                                            &
        OBJECTCONTROLS(NOLINES, NOCOMMENTS, NOPUBLICS,        &
                        NOSYMBOLS)
```

where:

input_file	Pathname of the link file produced previously by LINK86.
output_file	Pathname of the file in which LOC86 writes the absolute module.
stack_size	Size of this job's stack. Use this control for those jobs requiring a statically-allocated stack. If this control is required, specify a minimum value of 200H; otherwise specify zero.
rom_address	Absolute starting location of the PROM-resident class (code class) of the module.
ram_address	Absolute starting location of the RAM-resident classes of the module.
map_file	Pathname of the file in which LOC86 writes the locate map.

Use this form of the LOC86 command to locate each application first-level job and I/O job. The ORDER and ADDRESSES controls of this command differ from those of the RAM-based LOC86 command. In this command, the ORDER control does not mention the code class. The ADDRESSES control requires that you enter two absolute addresses; one to locate the code class in PROM and one to locate the remaining classes in RAM.

3.5.4 Testing the System in RAM

Before you actually locate a PROM/RAM system, it is recommended that you follow the procedures outlined in the previous sections, but specify RAM addresses for all classes. Then you can load the system into RAM and test it before burning code into PROM. After doing this, you can adjust the addresses to reflect a PROM/RAM system and build your final system.

ADDING USERS TO YOUR SYSTEM 4

4.1 INTRODUCTION

To function correctly, a system configured with the Human Interface requires information about all users (operators) and terminals that intend to access the system via the Human Interface. Two types of users exist for your system: a resident user and non-resident users.

4.2 THE RESIDENT USER

The resident user becomes part of your final system and resides in memory along with the rest of the operating system (thus, the term "resident user"). Two types of resident users exist: a recovery resident user and a non-recovery resident user. The recovery resident user gains control only if an initialization error occurs during initialization of the Human Interface. Regardless of the type, the resident user occupies one of the system terminals and is created before non-resident users. The operating system can contain information about only one resident user.

Including a resident user type in your system is called resident user configuration. Resident user configuration is accomplished by supplying information to the Human Interface (HI) screen during ICU configuration of the Human Interface. Refer to the *iRMX® I Interactive Configuration Reference Manual* for detailed information needed for resident user configuration.

4.3 NON-RESIDENT USERS

Non-resident users are users that can access the system using the Human Interface logon procedure. If your system is to be a multiple-user system, you need to define to the Human Interface all the non-resident users that can access the system. Configuration for non-resident users occurs through the Human Interface PASSWORD command and possible editing of several user definition files. These files define user names, limitations, passwords, terminals, and terminal characteristics to the system.

The process of adding non-resident users to your system is called non-resident user configuration. The files involved are called non-resident configuration files.

The system manager (who has user ID 0) can modify these files to add users or terminals, delete users or terminals, or change characteristics of users or terminals. Depending on the type of modifications made, the changes take effect either the next time the affected user logs onto the system or the next time the system is initialized. To prevent unauthorized users from changing the system configuration, the system manager should be the only user with change access to these files.

Refer to the *Operator's Guide to the iRMX® Human Interface* for detailed information on non-resident user configuration.

LOADING AND TESTING THE SYSTEM 5

5.1 INTRODUCTION

After you run the SUBMIT file generated by the ICU, you are ready to load the system into RAM and test it. The system RAM code is contained in the file that you specified while running the ICU. There are several different ways in which you can load your system into RAM.

5.2 LOADING YOUR SYSTEM INTO RAM

If you are using a Series IV development system, use the iSDM System Debug Monitor to load your system from disk into RAM. The iSDM monitor is described in the *iSDM™ System Debug Monitor Reference Manual.*

If you are using your System 300 Series Microcomputer as a development tool, use the Bootstrap Loader to load your system into RAM. The procedures for using the Bootstrap Loader are described in the *iRMX® Bootstrap Loader Reference Manual.*

5.3 INITIALIZING YOUR SYSTEM

After you load your system, you must initialize it. If you are using the Bootstrap Loader this process takes place automatically. If you did not load your system using the Bootstrap Loader, refer to the appropriate manual for instructions on how to initialize your system by starting execution from the beginning of the Root Job.

5.3.1 Initialization

An iRMX I Operating System can be configured to include your own code as a first-level job or as a first-level I/O job. When created, such a job contains only a single task. That single task creates or starts the creation of all other objects required by the first-level job. Thus it is referred to as the initialization task for its job, even though it may perform other functions as well. You should synchronize the operation of each initialization task with that of the root task to ensure proper functioning of your application system.

The root task is structured so that it creates the first-level jobs one at a time. It contains a programming loop that in general performs the following:

```
Repeat for each first-level job
    Create first-level job
    Suspend root task (until resumed by a first-level job)
Until finished
End
```

Each time the root task creates a first-level job, the root task suspends itself to allow the initialization task in the new job to perform synchronous initialization. Synchronous initialization consists of functions that must be performed immediately, before some other first-level job is created. Typically, this requires creating objects or making resources available that tasks in first-level jobs, not yet created, expect to be available when they themselves are created. (For example, the initialization task in the Extended I/O System job must create the entire Extended I/O System before it can allow the root task to create other first-level jobs that might make use of Extended I/O System functions.)

When the initialization task finishes its synchronous initialization, it must inform the root task that it is finished, so that the root task can resume execution and create another first-level job. The initialization task must always inform the root task that it has completed its synchronous initialization process by making the following procedure call:

```
CALL RQ$END$INIT$TASK;
```

This procedure call requires no parameters. When you call this procedure, the root task resumes execution, allowing it to create the next first-level job. You must include a call to RQ$END$INIT$TASK in the initialization task of each of your first-level jobs, even if the jobs require no synchronous initialization. If one of the first-level tasks does not include this call, the root job remains suspended and cannot create any of the remaining first-level jobs.

The amount of synchronous initialization that an initialization task must do depends on your job structure. You may require some of your initialization tasks to create all of the offspring jobs and a number of other objects before calling RQ$END$INIT$TASK. Some others may have to perform only one or two functions, call RQ$END$INIT$TASK, and then resume the process of initialization asynchronously. Still other initialization tasks may not have any synchronous initialization requirements and so can call RQ$END$INIT$TASK before performing any initialization. You must determine how the pieces of your system interact, and how they must be synchronized.

Another important factor in initialization is the order in which the root job creates the first-level jobs (see Table 5-1). The amount of processing your initialization tasks must do before calling RQ$END$INIT$TASK may depend on which jobs the root task has already created and which jobs it has yet to create. The order in which the root task creates first-level jobs depends on the order that you specify these jobs while running the ICU, not on the priority of the tasks in those jobs.

You should always use RQ$END$INIT$TASK as described in this section in order to perform your synchronous initialization. Otherwise, the root task cannot be resumed and thus, it cannot complete system initialization in the correct order.

Table 5-1. Order of Initialization

Order	First-Level Job
1	Root Job
2	Nucleus
3	System Debugger
4	Terminal Handler
5	Basic I/O System
6	Application Loader
7	Extended I/O System
8	I/O User Jobs
9	Universal Development Interface
10	User Jobs
11	Human Interface

5.3.2 System Initialization Errors

If the system encounters an error during the initialization process, it places diagnostic information in the processor registers and halts the processor. If the "Report Initialization Errors" entry on the Nucleus screen is "yes" and your processor board contains the iSDM monitor, a hexadecimal code and a mnemonic are displayed at the console indicating the layer that contains the initialization error.

Errors can occur during two operations:

- Nucleus and memory initialization
- Job creation by the root task

The value placed in the AX register determines which type of error occurred. The following sections outline these errors.

5.3.2.1 Nucleus and Memory Initialization Errors

If an error occurs during the memory initialization process, the Nucleus sets the processor registers as follows:

Register	Value	Description
AX	11H	A Nucleus or memory initialization error occurred. The BX register contains a description of the error if it occurred during memory initialization.
BX	0D001H	There are no memory blocks defined. There must be at least one.
	0D002H	Reserved.
	0D003H	Reserved.
	0D004H	There is not enough contiguous RAM available for the root job's memory pool.
	0D005H	Reserved.
	0D006H	There is not enough RAM available for the system resources of the Nucleus.
	0D007H	An invalid minimum transfer size was specified. Refer to the *iRMX® I Interactive Configuration Utility Reference Manual* for a description of the minimum transfer size.

5.3.2.2 Root Task Errors

If the root task encounters an error while it is creating the first-level jobs of your application system, it sets the processor registers as listed below. (If the "Report Initialization Errors" prompt is "Yes", it prints the error on the screen.)

Register	Value	Description
AX	21H	A root task error occurred. The BX, CX, and DL registers contain a description of the error.
BX	varies	BX is set as an index to indicate which first-level job caused the error. For example, 1 implies the first first-level job, 2 the second, and so forth.
CX	varies	CX contains the exception code returned from the CREATE$JOB system call that was called to create the first-level job.
DL	varies	DL contains the number of the parameter for the first-level job that caused the error. If DL is greater than 8, the parameter number is DL +1. Otherwise, the parameter number is DL.

5.3.2.3 System Debugger Initialization

The System Debugger defines a public symbol, RQ$SDB$INIT$ERROR (a WORD), in which it returns its initialization status. If the System Debugger initializes properly, it sets itself up as an operating system extension and sets RQ$SDB$INIT$ERROR to zero. If the System Debugger does not initialize correctly, it sets RQ$SDB$INIT$ERROR to a nonzero value.

If the "Report Initialization Errors" prompt is "Yes", errors will be written to the console. Otherwise, you must use the SDB.MP2 file to ascertain the address of the RQ$SDB$INIT$ERROR public symbol. Figure 5-1 shows the line of the SDB.MP2 locate map that contains this address. Once you have this address, use your monitor commands to find the initialization status of this system module.

```
                    •
                    •
                    •

    SYMBOL TABLE OF MODULE SBEGIN

    BASE    OFFSET TYPE SYMBOL          BASE   OFFSET TYPE SYMBOL

    36BFH   0000H  PUB  RQSDBINITTASK  4517H  0000H  PUB  RQSDBINITERROR

                    •
                    •
                    •
```

Figure 5-1. SDB.MP2 Locate Map

The initialization status codes returned by the system module are the same condition codes that the operating system could return for any exception condition.

5.3.2.4 Basic I/O System Initialization

The Basic I/O System defines a public symbol (an array), RQ$A$IOS$INIT$ERROR, in which it returns its initialization status. This array uses the following structure:

```
rq$a$ios$init$error STRUCTURE(
    Primitive$number        WORD
    Status                  WORD)
```

where the first error placed in the Primitive$number is the system call in the BIOS initialization that failed. If the Basic I/O System initializes properly, it sets itself up as an operating system extension, and sets RQ$A$IOS$INIT$ERROR to zero. If the Basic I/O System does not initialize correctly, it sets RQ$A$IOS$INIT$ERROR to a nonzero value.

If the "Report Initialization Errors" prompt is "Yes", errors will be written to the console. Otherwise, you must use the IOS.MP2 file to ascertain the address of RQ$A$IOS$INIT$ERROR public symbol. Once you have this address, use your monitor commands to find the initialization status of this system module. The initialization status codes returned by the system module are the same condition codes that the operating system could return for any exception condition.

5.3.2.5 Application Loader Initialization

The Application Loader defines a public symbol, RQ$LOADER$INIT$ERROR (a WORD), in which it returns its initialization status. If the Application Loader initializes properly, it sets itself up as an operating system extension and sets RQ$LOADER$INIT$ERROR to zero. If the Application Loader does not initialize correctly, it sets RQ$LOADER$INIT$ERROR to a nonzero value.

If the "Report Initialization Errors" prompt is "Yes", errors will be written to the console. Otherwise, you must use the LOADER.MP2 file to ascertain the address of RQ$LOADER$INIT$ERROR public symbol. Once you have this address, use your monitor commands to find the initialization status of this system module. The initialization status codes returned by the system module are the same condition codes that the operating system could return for any exception condition.

5.3.2.6 Extended I/O System Initialization

The Extended I/O System defines a public symbol, RQ$EIOS$INIT$ERROR (a WORD), in which it returns its initialization status. If the Extended I/O System initializes properly, it attaches all logical devices you specified with the ICU, sets itself up as an operating system extension, and sets RQ$EIOS$INIT$ERROR to zero. If the Extended I/O System does not initialize correctly, it sets RQ$EIOS$INIT$ERROR to a nonzero value.

If the "Report Initialization Errors" prompt, on the "Nucleus" screen, is "Yes", errors will be written to the console. Otherwise, you must use the EIOS.MP2 file to ascertain the address of RQ$EIOS$INIT$ERROR public symbol. Once you have this address, use your monitor commands to find the initialization status of this system module. The initialization status codes returned by the system module are the same condition codes that the operating system could return for any exception condition.

Once initialization is complete, users can create and attach files on the devices specified with the ICU. If the devices are off-line, an exceptional condition code is returned. If one of these devices is switched from on-line to off-line, the Extended I/O System automatically detaches the device, and all file connections on that device are marked invalid by the BIOS. When the unit is switched back on-line, the Extended I/O System automatically attaches it the first time a user tries to create or attach a file on the device. The Extended I/O System performs this service only for devices that it attaches.

5.3.2.7 Universal Development Interface Initialization

The Universal Development Interface defines a public symbol, RQ$UDI$INIT$ERROR (a WORD), in which it returns its initialization status. If the Universal Development Interface initializes properly, it sets itself up as an operating system extension, and sets RQ$UDI$INIT$ERROR to zero. If the Universal Development Interface does not initialize correctly, it sets RQ$UDI$INIT$ERROR to a nonzero value.

If the "Report Initialization Errors" prompt, on the "Nucleus" screen, is "Yes", the errors will be written to the console. Otherwise, you must use the UDI.MP2 file to ascertain the address of RQ$UDI$INIT$ERROR public symbol. Once you have this address, use your monitor commands to find the initialization status of this system module. The initialization status codes returned by the system module are the same condition codes that the operating system could return for any exception condition.

5.3.2.8 Human Interface Initialization

The Human Interface defines a public symbol, RQ$HI$INIT$ERROR (a WORD), in which it returns its initialization status. If the Human Interface initializes properly, it sets itself up as an operating system extension, and sets RQ$HI$INIT$ERROR to zero. If the Human Interface does not initialize correctly, it sets RQ$HI$INIT$ERROR to a nonzero value.

If the "Report Initialization Errors" prompt, on the "Nucleus" screen, is "Yes", the errors will be written to the console. Otherwise, you must use the HI.MP2 file to ascertain the address of RQ$HI$INIT$ERROR public symbol. Once you have this address, use your monitor commands to find the initialization status of this system module. The initialization status codes returned by the system module are the same condition codes that the operating system could return for any exception condition.

The only subsystem that handles an initialization error slightly different is the Human Interface. In addition to the initialization error described above, the Human Interface may issue the following warning if it does not have enough memory to fill the user's request.

```
*** WARNING:  THE SYSTEM DID NOT HAVE YOUR MINIMUM MEMORY REQUIREMENTS
              YOU WILL COME UP WITH ALL THE MEMORY THAT IS AVAILABLE IN THE
              SYSTEM, CONTACT THE SYSTEM MANAGER.
```

In such a case, the user is assigned whatever memory is available at the time.

5.4 TESTING YOUR SYSTEM

The normal development cycle is to load your system, test it and correct any errors, then reassemble or recompile any appropriate program code. Next, redefine and regenerate your system using the ICU, and load the system again. You can continue this procedure until you have created your target system. You can then copy your final system to PROM or use the Bootstrap Loader to load it from secondary storage.

If you are going to use the Bootstrap Loader to load your system, refer to *iRMX® Bootstrap Loader Reference Manual* for configuration information.

5.4.1 Using the Debugging Tools

The development of every system requires debugging and testing. To aid you in the development of iRMX I-based application systems, Intel provides the iRMX I Dynamic Debugger, the ICE-86A and I^2ICE In-Circuit Emulators, and the iRMX I System Debugger with the iSDM System Debug Monitor. The System Debugger extends the capabilities of the iSDM monitor. The following sections describe the advantages of these debugging tools.

5.4.1.1 Advantages of the iRMX® I Dynamic Debugger

The iRMX I Dynamic Debugger is a debugging tool that is "sensitive" to the data structures that the Nucleus maintains. The iRMX I Dynamic Debugger allows you to:

- Manipulate or examine any task while other tasks in the system continue to run. This distinguishes the iRMX I Dynamic Debugger from the iRMX I System Debugger, which requires that the application system be "frozen."
- Monitor system activity without interfering with execution.
- Examine and interpret data structures that are associated with the Nucleus and the Nucleus objects.

5.4.1.2 Advantages of the ICE™-86A and I^2ICE™ In-Circuit Emulator

The ICE™-86A and I^2ICE emulators provide in-circuit emulation for 8086 and 8088 microprocessor-based systems, meaning that it "stands in" for the these microprocessors in your target iRMX I-based system. The I^2ICE emulator also supports 80186, 80286, and 386™ microprocessor-based systems. The in-circuit emulators allow you to:

- Get closer to the hardware level by examining the contents of input pins and input ports.
- Change the values at output ports.
- Examine individual components rather than an entire board.
- Look at the most recent 80 to 150 assembly language instructions executed.

- Protect memory areas from being altered and trap on attempted access.

5.4.1.3 Advantages of the iRMX® I System Debugger

You can extend the capabilities of the iSDM monitor by including the System Debugger as part of your operating system. In addition to retaining the features of the monitor, the System Debugger

- Identifies and interprets iRMX I system calls.
- Displays iRMX I objects.
- Allows the user to examine the stack of a task to determine which iRMX I system calls it has made recently.

5.4.2 Debugging Application Jobs

While you are creating your application jobs, you will probably use the following iterative procedure to remove bugs from your code:

1. Configure your system.
2. Generate your system using ICU generated command files.
3. Test the system to find bugs.
4. If any bugs are found, modify the application code to eliminate the bugs and go to Step 2.

To remove most of the bugs from your application software. you might have to loop several times through these three steps. The purpose of this section is to show you how to simplify the process of configuring your system during development. By using the techniques presented here, you can reduce the time you spend in configuration and increase the time available for debugging.

5.4.2.1 Summary of Configuration

Configuration is a three-phase process:

1. Using the ICU to select the iRMX I software that meets the needs of your application.
2. Decide where in memory to place your code modules and data segments, then link and locate the code and data.
3. Tell the ICU where the code and data are located.

Once you have performed these three phases, you need only load the code and start up the root job in order to get the entire system running.

5.4.2.2 Configuration and Debugging

During the process of debugging an application, you generally perform Phase 1 of configuration only once, and Phases 2 and 3 repeatedly. You need not repeat Phase 1 because your application generally uses the same set of iRMX I system calls throughout debugging. On the other hand, Phases 2 and 3 are generally repeated because the application software modules change frequently during debugging.

By using a special method during the coding of your initial task software, you can freeze the locations of your application software modules and data segments. This reduces the probability of your repeating Phases 2 and 3 of the configuration process.

5.4.2.3 The Technique

You must specify values for three parameters in the "I/O User Jobs" screen and the "User Jobs" screen that are very volatile during development. These parameters are "(AEH) Address of Exception Handler", "(TSA) Task Start Address", and "(DSB) Data Segment Base".

During debugging, as you modify code and (consequently) change the size of your code modules, the values that you must assign to these three parameters are very likely to change. By heeding the following two suggestions, you can significantly reduce the likelihood of changing these parameters and, hence, you can retest your revised application job after merely linking and loading.

5.4.2.4 Freezing the Base of the Data Segment

If, during development, you locate your job's data segment after your job's code segment, you can freeze the base of the data segment by padding the code segment. Consider the following two situations.

In Job A (Figure 5-2), the code modules are located contiguously, with the data segment immediately following the last module. If any of the modules in Job A grow or shrink as a result of debugging, you must relocate the data segment. This involves changing the "(DSB) Data Segment Base" parameter for the job and regenerating the system.

In contrast, Job B (Figure 5-2) is designed to accommodate modification. The modules are still located contiguously, but some unused memory has been left between the code segment and the data segment. This unused memory, called padding, allows the modules in the code segment to grow without causing a change in the base address of the data segment.

You must decide how much padding to leave between the code and data segments. In general, the less stable the code is, the more padding you should leave. If you are uncertain, try starting with 1000 bytes.

In order to obtain the padding between the code and data segments, you can use the address control of the LOC86 command. For example,

```
ADDRESSES(CLASSES(CODE(aaaaa), DATA(bbbbb)))
```

where aaaaa is the address at which you want to place the job's code segment, and bbbbb is the address at which you want to place the job's data segment. You can compute bbbbb by adding the size of the padding to the address of the end of the code segment. It is also a good idea to pad the data segment.

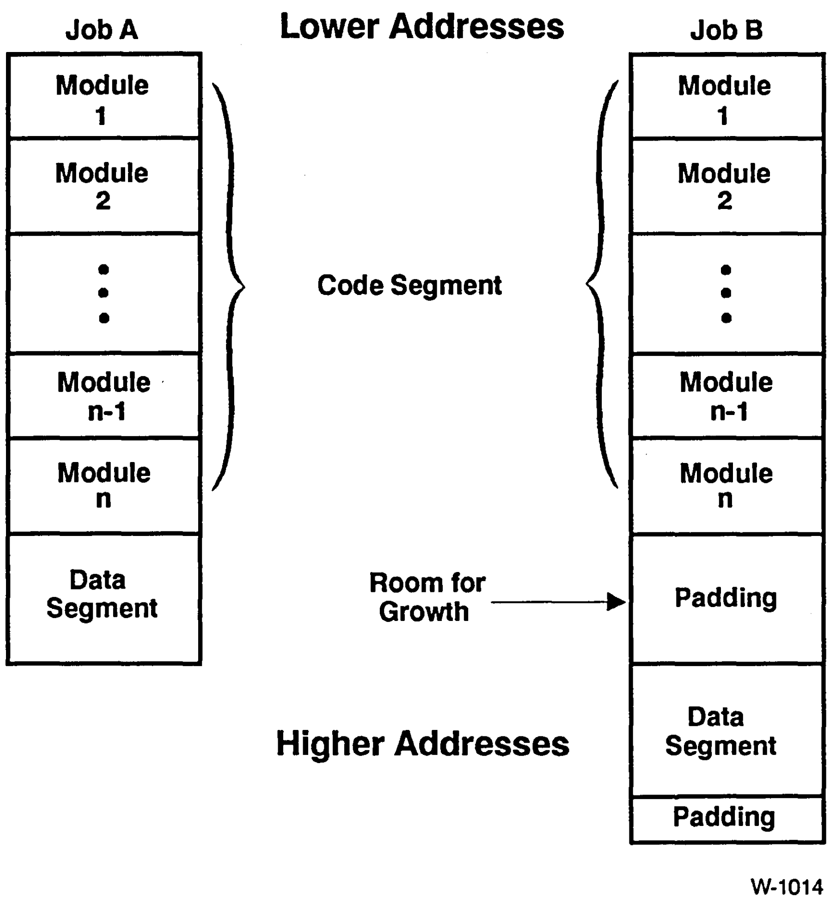


Figure 5-2. How To Freeze The Base Of The Data Segment

5.4.2.5 Freezing the Entry Points

The ICU requires the addresses of two entry points, one for the job's initial task, and one for the job's exception handler. Because these addresses are expressed as offsets from the base of the job's code segment, you can freeze the addresses by preventing the offsets from changing.

The easiest way to accomplish this is to create a special module that contains new entry points for the initial task and the exception handler. This special module, if located at the front of your code segment, provides entry points that are completely independent of changes made to other modules.

Within this special module, each entry point must be coded as a procedure containing only a procedure call followed by a return instruction. The purpose of the procedure call is to invoke a secondary, external procedure that actually contains the initial task or the exception handler. Figure 5-3 illustrates the special module in pseudo-code.

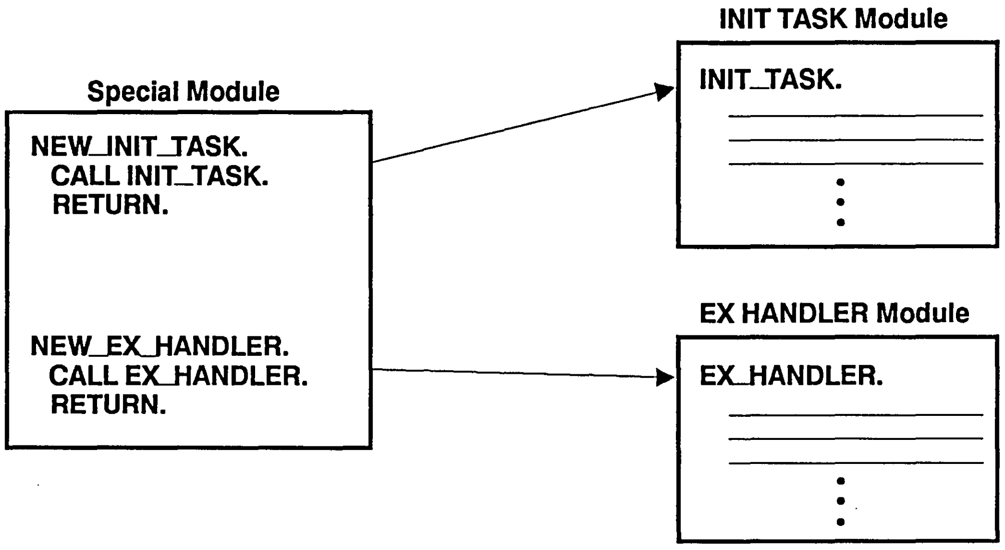


Figure 5-3. Special Module Freezes Entry Points

You can place the special module at the front of your code segment (Figure 5-4) by linking it first during the linking process. This will ensure that the new entry points for the initial task and the exception handler are ahead of the code modules that are subject to change. This, in turn, ensures that the new entry points will remain a fixed distance from the base of the code segment, and that you will not need to modify the exception_handler_entry or the init_task_entry parameters.

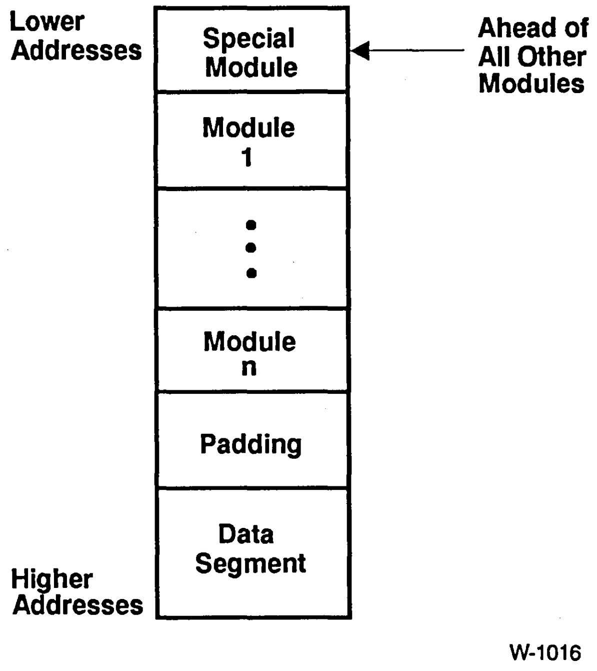


Figure 5-4. Location Of The Special Module

FILES CREATED BY THE ICU A

A.1 INTRODUCTION

The files listed in this appendix are created/recreated by the ICU or as output of the SUBMIT file.

A.2 CREATED FILES

The table below lists the files that are created/recreated whenever you issue the G command from the ICU and/or invoke the ICU-generated SUBMIT file. The file names listed here do not include the prefix letter that may be added before generation.

Table A-1. Files Created by the ICU and SUBMIT File

Subsystem	Created by ICU	Created by SUBMIT File
Nucleus	NTABL.A86	NTABL.OBJ NTABL.LST
	NDEVC.A86	NDEVC.OBJ NDEVC.LST NUCLS NUCLS.LNK NUCLS.MP1 NUCLS.MP2
Terminal Handler	MTHn.A86 where n = 1 to 7	MTHn.OBJ MTHn.LST MTHn.LNK MTHn.MP1

Table A-1. Files Created by the ICU and SUBMIT File (continued)

Subsystem	Created by ICU	Created by SUBMIT File
Basic I/O System	ICDEV.A86	ICDEV.OBJ ICDEV.LST
	ITABL.A86	ITABL.OBJ ITABL.LST
	ITDEV.A86	ITDEV.OBJ ITDEV.LST IOS IOS.LNK IOS.MP1 IOS.MP2 IOS1.LNK IOS1.MP1 TSC.LNK TSC.MP1
Extended I/O System	EDEVC.A86	EDEVC.OBJ EDEVC.LST
	ETABL.A86	ETABL.OBJ ETABL.LST
	EJOBC.A86	EJOBC.OBJ EJOBC.LST EIOS EIOS.LNK EIOS.MP1 EIOS.MP2
Application Loader	LCONF.P86	LCONF.OBJ LCONF.LST LOADR LOADR.LNK LOADR.MP1 LOADR.MP2
Human Interface	HCONF.P86	HCONF.OBJ HCONF.LST HCLI.LNK HCLI.MP1 HI HI.LNK HI.MP1 HI.MP2

Table A-1. Files Created by the ICU and SUBMIT File (continued)

Subsystem	Created by ICU	Created by SUBMIT File
UDI	UDICN.A86	UDICN.OBJ UDICN.LST UDI UDI.LNK UDI.MP1 UDI.MP2
Root Job	ROOT.A86	ROOT.OBJ ROOT.LST ROOT CROOT.LNK CROOT.MP1 ROOT.MP2
System Debugger	SDBCN.A86	SDBCN.OBJ SDBCN.LST SDB.LNK SDB.MP1
Others	<output-file>.CSD ICULOC.CSD ICUROT.CSD	boot-loadable iRMX I file <bootloadable-file>.MP2

EXAMPLE SYSTEM CONFIGURATION B

B.1 INTRODUCTION

This appendix contains an example illustrating how to use the ICU to modify an Intel-supplied definition file. This example contains the following descriptions:

- The configuration defined by the Intel-supplied definition file (28612.DEF).
- The target system, focusing on the differences between it and the supplied configuration.
- The ICU changes required to convert the existing definition file to one corresponding to the target system.

B.2 THE INTEL-SUPPLIED DEFINITION FILE

The existing definition file, named 28612.DEF, defines the 80286-based multi-user system. This particular system configuration has the following characteristics:

- The CPU board is an iSBC 286/10(A) or an iSBC 286/12 board.
- Master Interrupt levels are assigned as follows:

  Level 0 - System Clock

  Level 1 - System Debugger

  Level 2 - Available

  Level 3 - Used by the Terminal Communications Controller

  Level 4 - Available

  Level 5 - An MSC controller

  Level 6 - An 8274 Terminal Driver

  Level 7 - An 8259A slave PIC

- Up to 896K bytes of RAM, at addresses 0H through 0DFFF:FH. Of these, addresses 120:0H through 0DFFF:FH are free for use by the system.
- The system device is :SD:.
- The supplied, ready-to-bootstrap-load file is /BOOT86/28612.

B.3 DIFFERENCES BETWEEN THE TARGET AND START-UP SYSTEMS

The differences between the target system and the 80286-based multi-user system dictate how you will use the ICU to alter the definition file. These systems differ in the following ways:

- The iSBX 350 Parallel MULTIMODULE is an addition to the target system. This board controls a Centronics interface line printer.
- The iSBX 351 Serial MULTIMODULE is an addition to the target system. This board controls one RS232C serial channel when used with the 8251A Terminal Driver.
- The iSBC 544A Intelligent Communication Controller on the target system uses interrupt level 48H, instead of interrupt level 38H.
- The iSBC 208 flexible disk controller is not part of the target system.
- The target system resides in a bootloadable file named /BOOT86/SAM86.

The name of the new definition file for the target system is SAM86.DEF.

B.4 STEPS PERFORMED TO CREATE THE TARGET SYSTEM

The steps needed to modify an existing definition file to meet the target system needs are outlined below.

- Add the 8251A Terminal Driver (for the iSBX 351 board).
- Add the iSBX 350 Line Printer Driver.
- Change the interrupt level for the iSBC 544A Intelligent Communication Controller.
- Remove the iSBC 208 Driver.
- Change the name of the resulting bootloadable file

As you proceed through this example refer to the *iRMX® I Interactive Configuration Utility Reference* manual for more information about configuring each of the above drivers.

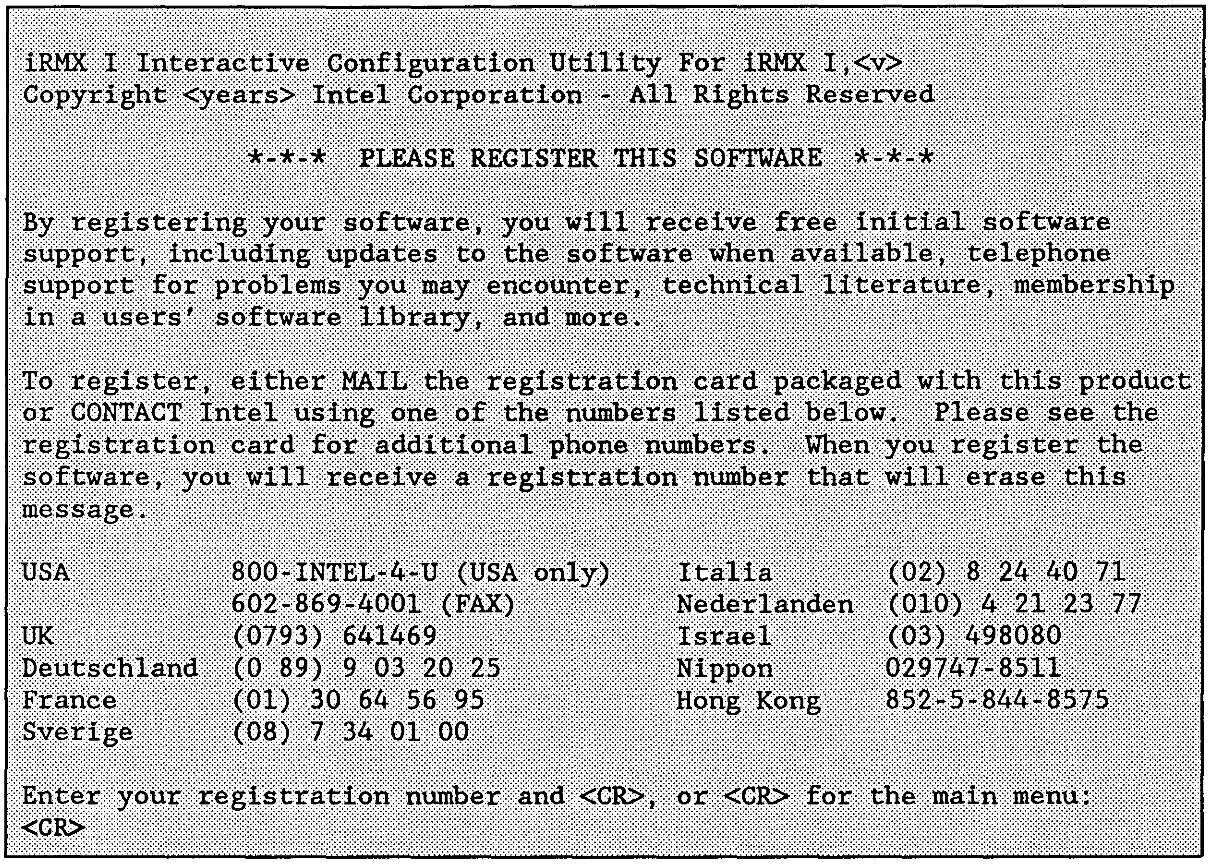
B.5 USING THE ICU TO DEFINE THE TARGET SYSTEM

This section describes a dialogue between a user and the ICU. This dialogue demonstrates the steps needed to define the target system described in the previous section. In the dialogue, user input is shown in either blue or bold text, followed by a carriage return (<CR>). Should you make an error in entering information as you proceed through this example, you can re-type the information if you are currently viewing the screen in which the error was entered. If not, you can use either the Backup (b) or Find (f) command to access the screen you want to change, then re-type the correct information. If you are new to the iRMX I ICU and do not want to use these commands, you can delete the SAM86.DEF file in the SAM86 directory and start over by entering the last line of the command sequence listed below.

Invoke the ICU, giving the name of the default file and the desired name of the modified definition file, as follows:

```
- CREATEDIR sam86 <CR>
- ATTACHFILE sam86 <CR>
- ICU86 :RMX:ICU/28612.def to sam86.def <CR>
```

This produces the registration message screen shown in Figure B-1. The registration screen appears each time you invoke the ICU until you obtain your registration number and enter it at the prompt line at the bottom of the screen.



```
iRMX I Interactive Configuration Utility For iRMX I,<v>
Copyright <years> Intel Corporation - All Rights Reserved

              *-*-*  PLEASE REGISTER THIS SOFTWARE  *-*-*

By registering your software, you will receive free initial software
support, including updates to the software when available, telephone
support for problems you may encounter, technical literature, membership
in a users' software library, and more.

To register, either MAIL the registration card packaged with this product
or CONTACT Intel using one of the numbers listed below.  Please see the
registration card for additional phone numbers.  When you register the
software, you will receive a registration number that will erase this
message.

USA          800-INTEL-4-U (USA only)    Italia        (02) 8 24 40 71
             602-869-4001 (FAX)          Nederlanden   (010) 4 21 23 77
UK           (0793) 641469               Israel        (03) 498080
Deutschland  (0 89) 9 03 20 25           Nippon        029747-8511
France       (01) 30 64 56 95            Hong Kong     852-5-844-8575
Sverige      (08) 7 34 01 00

Enter your registration number and <CR>, or <CR> for the main menu:
<CR>
```

Figure B-1. Registration Message Screen

When you press <CR>, the main menu screen is displayed as follows:

```
For general help in any screen enter H <CR>.

The following commands are available

Change
Generate
List
Save
Quit
Exit
Replace
Detail-Level
Backup

 ENTER COMMAND : C idevs  <CR>
```

Figure B-2. Main Menu Screen

The first change in the target system is to add the iSBX 350 Line Printer Driver. To do this, go to the "Intel Device Drivers" screen by entering "C idevs <CR>", as shown in Figure B-2. This produces the screen shown in Figure B-3.

```
(IDEVS)      Intel Device Drivers

(S14) Mass Storage Controller Driver [Yes/No]   YES
(T74) 8274 Terminal Driver  [Yes/No]            YES
(T51) 8251A Terminal Driver [Yes/No]            NO
(T30) 82530 Terminal Driver [Yes/No]            NO
(TCC) Terminal Comm Controller [Yes/No]         YES
(L86) Line Printer - iSBC 286/10 [Yes/No]       YES
(L50) Line Printer - iSBX 350 [Yes/No]          NO
(S20) iSBC 220         [Yes/No] NO  (X18) iSBX 218A      [Yes/No] NO
(S08) iSBC 208         [Yes/No] YES (S54) iSBC 254       [Yes/No] YES
(T34) iSBC 534         [Yes/No] NO  (T44) iSBC 544A      [Yes/No] YES
(RAM) RAM Disk Driver  [Yes/No] NO  (SCS) SCSI Driver    [Yes/No] NO
(S64) iSBC 264         [Yes/No] NO  (THD) Term. Handler  [Yes/No] NO

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: L50=y <CR>
```

Figure B-3. The Intel Device Drivers Screen

The "Intel Device Drivers" screen, shown in Figure B-3, lists all of the available Intel devices. The "YES" or "NO" field shown to the right of each device indicates whether or not it is part of the current definition file. To add a device or go to the first screen of an existing device, enter its three-letter abbreviation followed by "=y <CR>". The abbreviation for the iSBX 350 is "L50", so type "L50=Y <CR>". This produces the screen shown in Figure B-4.

```
Do you want any/more  Line Printer - iSBX 350 DEVICEs ?
y <CR>
```

Figure B-4. Query Screen for the iSBX 350 Device

Figure B-4 shows a query screen that asks if you want to add a device. To start the process of adding the iSBX 350 MULTIMODULE Driver, type "y <CR>" as shown in Figure B-4. This will produce the "Line Printer - iSBX 350" screen, as shown in Figure B-5.

```
(D350)       Line Printer - iSBX 350

(DEV) Device Name [1-16 Chars]
(IL)  Interrupt Level [Encoded Level]            073H
(ITP) Interrupt Task Priority [0-255]            130
(POA) 8255A Port A Address [0-0FFFFH]            080H
(POB) 8255A Port B Address [0-0FFFFH]            082H
(POC) 8255A Port C Address [0-0FFFFH]            084H
(CON) 8255A Control Port Address [0-0FFFFH]      086H
(TAB) Printer Expands Tabs [Yes/No]              YES
(ITO) Interrupt Time Out [0-0FFFFH]              0200H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: dev=d350 <CR>
: il=75h <CR>
: poa=a0h <CR>
: pob=a2h <CR>
: poc=a4h <CR>
: con=a6h <CR>
: <CR>
```

Figure B-5. Line Printer - iSBX™ 350 Device Screen

On this screen you need to enter a device name, change the interrupt level, and change the 8255A port addresses to match the hardware configuration. The commands to make these changes are shown in Figure B-5. Typing a "<CR>" by itself, as shown at the bottom of Figure B-5, will redisplay the screen with the changes completed. This produces the screen shown in Figure B-6.

```
(D350)        Line Printer - iSBX 350

(DEV) Device Name [1-16 Chars]                    D350
(IL)  Interrupt Level [Encoded Level]             075H
(ITP) Interrupt Task Priority [0-255]             130
(POA) 8255A Port A Address [0-0FFFFH]             0A0H
(POB) 8255A Port B Address [0-0FFFFH]             0A2H
(POC) 8255A Port C Address [0-0FFFFH]             0A4H
(CON) 8255A Control Port Address [0-0FFFFH]       0A6H
(TAB) Printer Expands Tabs [Yes/No]               YES
(ITO) Interrupt Time Out [0-0FFFFH]               0200H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: <CR>
```

Figure B-6. Completed Line Printer - iSBX™ 350 Device Screen

Check that you entered the changes correctly. If so, you are ready to add the iSBX 350 device unit information. Typing a "<CR>" by itself, as shown in Figure B-6, will display the query screen in Figure B-7.

```
Do you want any/more  Line Printer - iSBX 350 DEVICEs ?
<CR>
```

Figure B-7. Query Screen for another iSBX™ 350 Device

This example requires only one iSBX MULTIMODULE, so respond with a carriage return (<CR>) as shown in Figure B-7. This tells the ICU that you do not want another iSBX 350 Driver and displays the next screen, as shown in Figure B-8.

```
Do you want any/more Line Printer - iSBX 350 DUIBs ?
y <CR>
```

Figure B-8. Query Screen for iSBX™ 350 Device-Unit Information

Figure B-8 shows a query screen that asks if you want to fill in Device-Unit Information for an iSBX 350 MULTIMODULE. This is the first time that an iSBX 350 MULTIMODULE has been added, so such information does not yet exist. Respond to this screen by entering "y <CR>", as shown in Figure B-8. This produces the "Line Printer - iSBX 350 Device-Unit Information" screen shown in Figure B-9.

```
(I350)    Line Printer - iSBX 350 Device-Unit Information

(DEV) Device Name [1-16 Chars]
(NAM) Device-Unit Name [1-14 Chars]
(MB)  Max Buffers [0-0FFH]              0H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: dev=d350 <CR>
: nam=lp350 <CR>
: <CR>
```

Figure B-9. Line Printer - iSBX™ 350 Device-Unit Information Screen

Notice that the "(DEV) Device Name" and "(NAM) Device-Unit Name" fields are blank. These fields must be filled in.

To fill in the first field, enter "dev=d350 <CR>" as shown in Figure B-9. This is the same device name you specified on the "Line Printer - iSBX 350 Device Screen." By entering the same device name, the information on both screens is used to configure the same device.

For the second field, enter "nam=lp350 <CR>", as shown in Figure B-9. Finally, enter a carriage return (<CR>) by itself to redisplay the screen with the changes you just made. This produces the screen shown in Figure B-10.

```
(I350)    Line Printer - iSBX 350 Device-Unit Information

(DEV) Device Name [1-16 Chars]          D350
(NAM) Device-Unit Name [1-14 Chars]     LP350
(MB)  Max Buffers [0-0FFH]              0H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: f idevs <CR>
```

Figure B-10. Completed Line Printer - iSBX™ 350 Device-Unit Information Screen

If all of the entries are correct, all of the steps to include the iSBX 350 Driver are completed.

The next step is to add the 8251A Terminal Driver to support the iSBX 351 Serial MULTIMODULE. To begin, return to the "Intel Device Drivers" screen by entering "f idevs <CR>" as shown in Figure B-10.

To add any driver from the (IDEVS) screen, you type the device's three-letter abbreviation and "=y <CR>". The abbreviation for the 8251A Terminal Driver is "T51", so you type "T51=y <CR>" as shown in Figure B-10. This produces the screen shown in Figure B-11.

```
Do you want any/more 8251A Terminal Driver DEVICEs ?
y <CR>
```

Figure B-11. Query Screen for 8251A Terminal Driver

Figure B-11 shows a query screen that asks if you want to add a device. To start the process of adding the 8251A Terminal Driver, type "y <CR>" as shown in Figure B-11. This will produce the "8251A Terminal Driver" screen, as shown in Figure B-12.

```
(D8251)      8251A Terminal Driver

(DEV) Device Name [1-16 Chars]
(IIL) Input Interrupt Level [Encoded]        073H
(OIL) Output Interrupt Level [Encoded]       074H
(UDP) USART Data Port [0-0FFFFH]             080H
(USP) USART Status Port [0-0FFFFH]           082H
(IRP) 8254 Inrate Port [0-0FFFFH]            094H
(ICP) 8254 Input Control Port [0-0FFFFH]     096H
(IRC) 8254 Input Counter Number [0-2]        2
(IRF) Inrate Frequency [0-0FFFFFFFFH]        012C000H
(ORP) 8254 Outrate Port [0-0FFFFH]           0H
(OCP) 8254 Output Control Port [0-0FFFFH]    0H
(ORC) 8254 Output Counter Number [0-2]       0
(ORF) Outrate Frequency [0-0FFFFFFFFH]       0H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: dev=d8251 <CR>
: <CR>
```

Figure B-12. 8251A Terminal Driver Screen

The default values on this screen match the target system. The only field that needs to be filled in is "(DEV) Device Name". To fill in this field, enter "dev=d8251 <CR>" as shown in Figure B-12. To redisplay the screen so you can check your entries, enter a carriage return (<CR>) by itself as shown in Figure B-12. This produces the screen shown in B-14.

```
(D8251)       8251A Terminal Driver

(DEV) Device Name [1-16 Chars]               D8251
(IIL) Input Interrupt Level [Encoded]        073H
(OIL) Output Interrupt Level [Encoded]       074H
(UDP) USART Data Port [0-0FFFFH]             080H
(USP) USART Status Port [0-0FFFFH]           082H
(IRP) 8254 Inrate Port [0-0FFFFH]            094H
(ICP) 8254 Input Control Port [0-0FFFFH]     096H
(IRC) 8254 Input Counter Number [0-2]        2
(IRF) Inrate Frequency [0-0FFFFFFFFH]        012C000H
(ORP) 8254 Outrate Port [0-0FFFFH]           0H
(OCP) 8254 Output Control Port [0-0FFFFH]    0H
(ORC) 8254 Output Counter Number [0-2]       0
(ORF) Outrate Frequency [0-0FFFFFFFFH]       0H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: <CR>
```

Figure B-13. Completed 8251A Terminal Driver Screen

Check that you have entered the device name correctly. If so, you are ready to add the 8251A unit information. To tell the ICU you are ready to view the next screen, enter a carriage return (<CR>) by itself as shown in Figure B-13. This produces the screen shown in Figure B-14.

```
Do you want any/more 8251A Terminal Driver DEVICEs ?
<CR>
```

Figure B-14. Query Screen for another 8251A Terminal Driver

This example requires only one 8251A Terminal Driver, so you should respond by entering a carriage return (<CR>) as shown in Figure B-14. This tells the ICU that you do not want another 8251A Terminal Driver and causes the "8251A Terminal Driver UNITs" query screen to appear.

```
Do you want any/more 8251A Terminal Driver UNITs ?
y <CR>
```

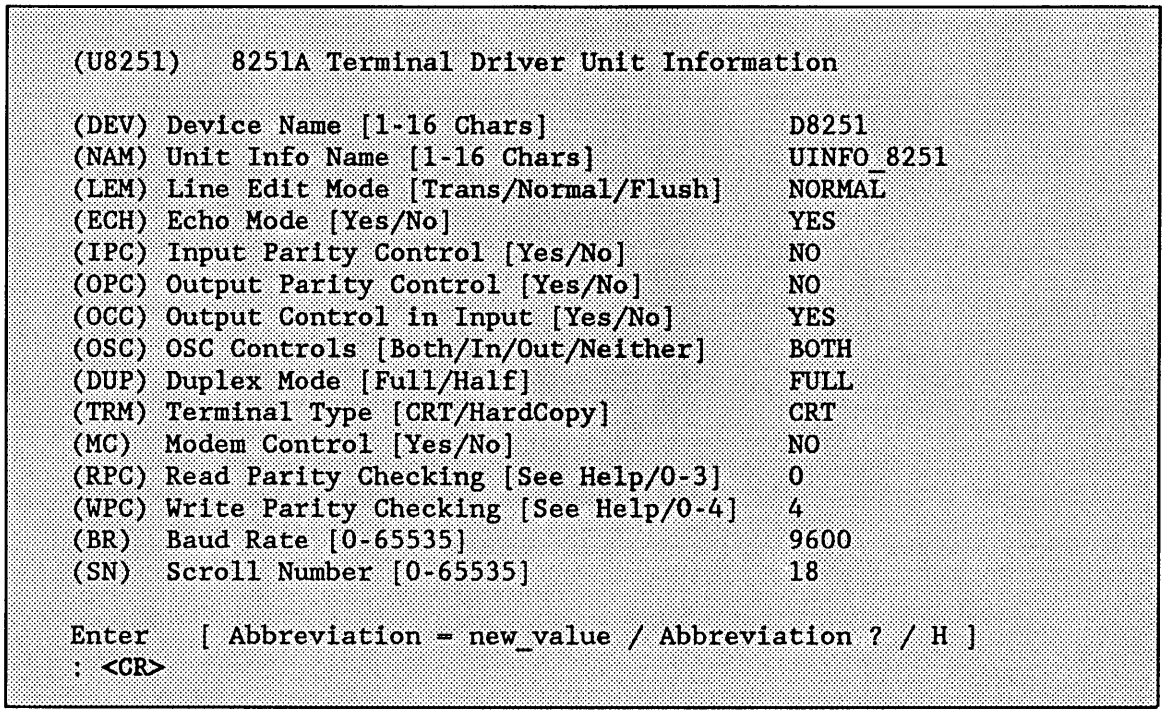
Figure B-15. Query Screen for 8251A Terminal Driver Units

Because this is a newly added device, you must complete all screens related to the 8251A Terminal Driver. Respond to the query screen by entering "y <CR>", as shown in Figure B-15 to display the screen shown in Figure B-16.

```
(U8251)    8251A Terminal Driver Unit Information

(DEV) Device Name [1-16 Chars]
(NAM) Unit Info Name [1-16 Chars]
(LEM) Line Edit Mode [Trans/Normal/Flush]     NORMAL
(ECH) Echo Mode [Yes/No]                      YES
(IPC) Input Parity Control [Yes/No]           NO
(OPC) Output Parity Control [Yes/No]          NO
(OCC) Output Control in Input [Yes/No]        YES
(OSC) OSC Controls [Both/In/Out/Neither]      BOTH
(DUP) Duplex Mode [Full/Half]                 FULL
(TRM) Terminal Type [CRT/HardCopy]            CRT
(MC)  Modem Control [Yes/No]                  NO
(RPC) Read Parity Checking [See Help/0-3]     0
(WPC) Write Parity Checking [See Help/0-4]    4
(BR)  Baud Rate [0-65535]                     9600
(SN)  Scroll Number [0-65535]                 18

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: dev=d8251 <CR>
: nam=uinfo_8251 <CR>
: <CR>
```

Figure B-16. 8251A Terminal Driver Unit Information Screen

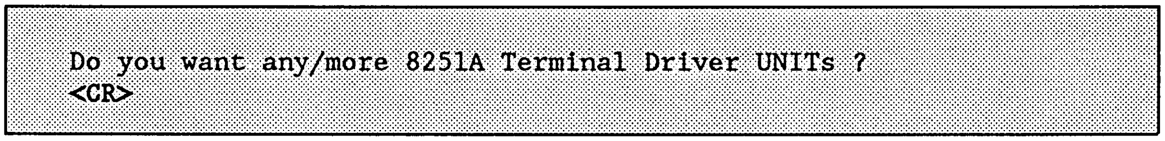
Again, the default values on this screen match the values required by the target system. However, the first two fields are blank and must be filled in. Fill in the "(DEV) Device Name" field by entering "dev=d8251 <CR>" as shown in Figure B-16. For the second field, "(NAM) Unit Info Name", enter "nam=uinfo_8251 <CR>". Finally, enter a carriage return (<CR>) by itself to redisplay the screen and check the values you just entered. This produces the screen shown in Figure B-17.



```
(U8251)   8251A Terminal Driver Unit Information

(DEV) Device Name [1-16 Chars]                 D8251
(NAM) Unit Info Name [1-16 Chars]              UINFO_8251
(LEM) Line Edit Mode [Trans/Normal/Flush]      NORMAL
(ECH) Echo Mode [Yes/No]                       YES
(IPC) Input Parity Control [Yes/No]            NO
(OPC) Output Parity Control [Yes/No]           NO
(OCC) Output Control in Input [Yes/No]         YES
(OSC) OSC Controls [Both/In/Out/Neither]       BOTH
(DUP) Duplex Mode [Full/Half]                  FULL
(TRM) Terminal Type [CRT/HardCopy]             CRT
(MC)  Modem Control [Yes/No]                   NO
(RPC) Read Parity Checking [See Help/0-3]      0
(WPC) Write Parity Checking [See Help/0-4]     4
(BR)  Baud Rate [0-65535]                      9600
(SN)  Scroll Number [0-65535]                  18

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: <CR>
```

Figure B-17. Completed 8251A Terminal Driver Unit Information Screen

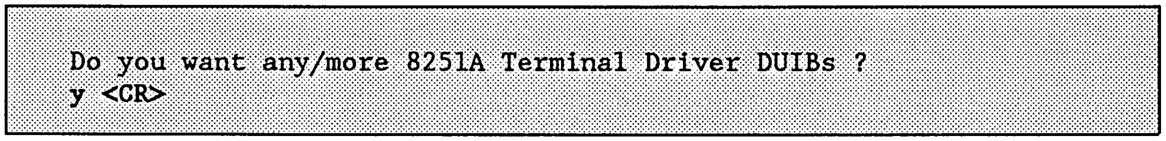
When you have confirmed that the values you entered are correct, enter a carriage return (<CR>) as shown in Figure B-17. This produces the query screen shown in Figure B-18.



```
Do you want any/more 8251A Terminal Driver UNITs ?
<CR>
```

Figure B-18. Query Screen for another 8251A Terminal Driver Unit

This application requires only one 8251A unit. To continue completing the screens related to the 8251A Terminal Driver, enter a carriage return (<CR>), as shown in Figure B-18. This displays the query screen shown in Figure B-19.



```
Do you want any/more 8251A Terminal Driver DUIBs ?
y <CR>
```

Figure B-19. Query Screen for 8251A Terminal Driver DUIBs

To display the last screen you need to fill in for the 8251A Terminal Driver, enter "y <CR>" as shown in Figure B-19. This produces the screen shown in Figure B-20.

```
(I8251)   8251A Terminal Driver Device-Unit Information

(DEV) Device Name [1-16 Chars]
(NAM) Device-Unit Name [1-14 Chars]
(UN)  Unit Number on this Device [0-0FFH]    0H
(UIN) Unit Info Name [1-16 Chars]
(MB)  Max Buffers [0-0FFH]                   0H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: dev=d8251 <CR>
: nam=t351_0 <CR>
: uin=uinfo_8251 <CR>
: <CR>
```

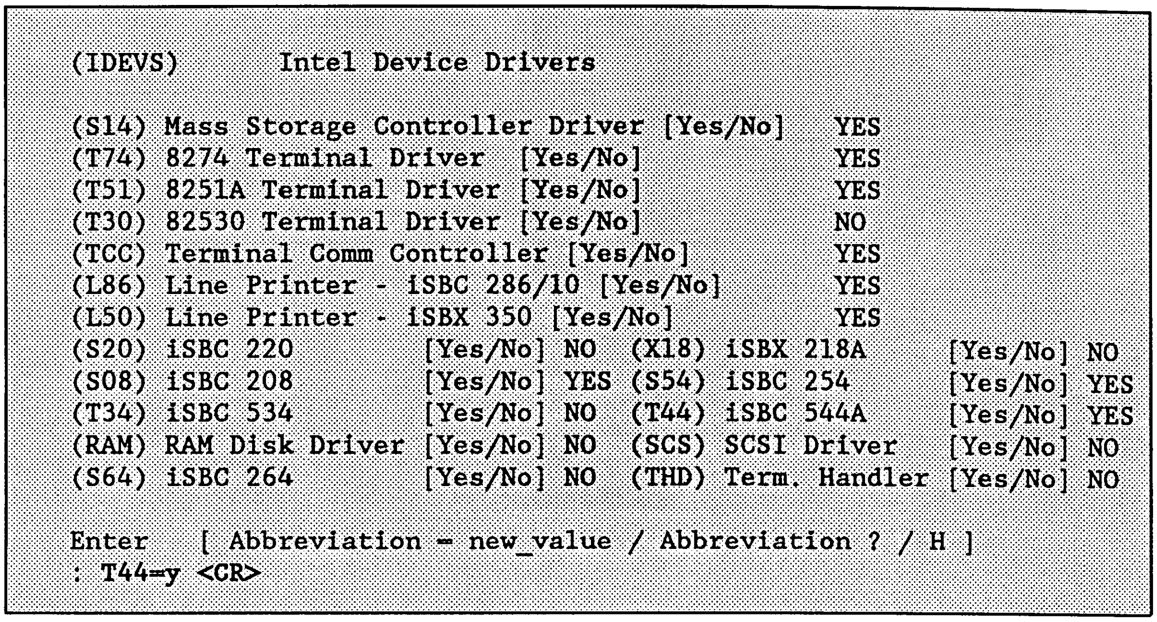
Figure B-20. 8251A Terminal Driver Device-Unit Information Screen

The fields with default values on this screen match the target system, but the three blank fields must be filled in. Enter "dev=d8251 <CR>", "nam=t351_0 <CR>", and "uin=uinfo_8251 <CR>" as shown in Figure B-20. Redisplay the screen to check your entries by entering a carriage return (<CR>).

```
(I8251)   8251A Terminal Driver Device-Unit Information

(DEV) Device Name [1-16 Chars]               D8251
(NAM) Device-Unit Name [1-14 Chars]          T351_0
(UN)  Unit Number on this Device [0-0FFH]    0H
(UIN) Unit Info Name [1-16 Chars]            UINFO_8251
(MB)  Max Buffers [0-0FFH]                   0H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: f idevs <CR>
```

Figure B-21. Completed 8251A Terminal Driver Device-Unit Information Screen

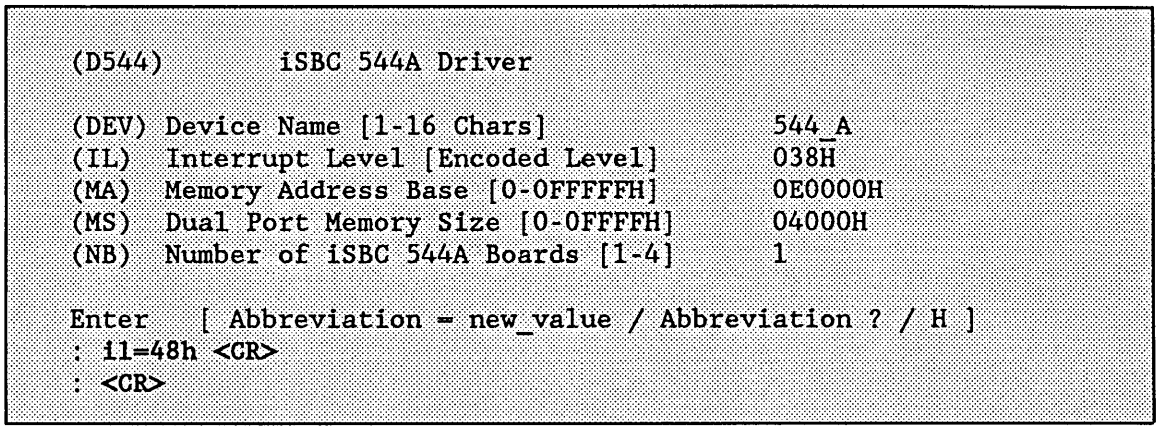
All of the steps to include the 8251A Terminal Driver are completed. Now, return to the "Intel Device Drivers" screen by entering "f idevs <CR>" as shown in Figure B-21.



```
(IDEVS)      Intel Device Drivers

(S14) Mass Storage Controller Driver [Yes/No]    YES
(T74) 8274 Terminal Driver  [Yes/No]             YES
(T51) 8251A Terminal Driver [Yes/No]             YES
(T30) 82530 Terminal Driver [Yes/No]             NO
(TCC) Terminal Comm Controller [Yes/No]          YES
(L86) Line Printer - iSBC 286/10 [Yes/No]        YES
(L50) Line Printer - iSBX 350 [Yes/No]           YES
(S20) iSBC 220         [Yes/No] NO  (X18) iSBX 218A     [Yes/No] NO
(S08) iSBC 208         [Yes/No] YES (S54) iSBC 254      [Yes/No] YES
(T34) iSBC 534         [Yes/No] NO  (T44) iSBC 544A     [Yes/No] YES
(RAM) RAM Disk Driver  [Yes/No] NO  (SCS) SCSI Driver   [Yes/No] NO
(S64) iSBC 264         [Yes/No] NO  (THD) Term. Handler [Yes/No] NO

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: T44=y <CR>
```

Figure B-22. Intel Device Drivers Screen

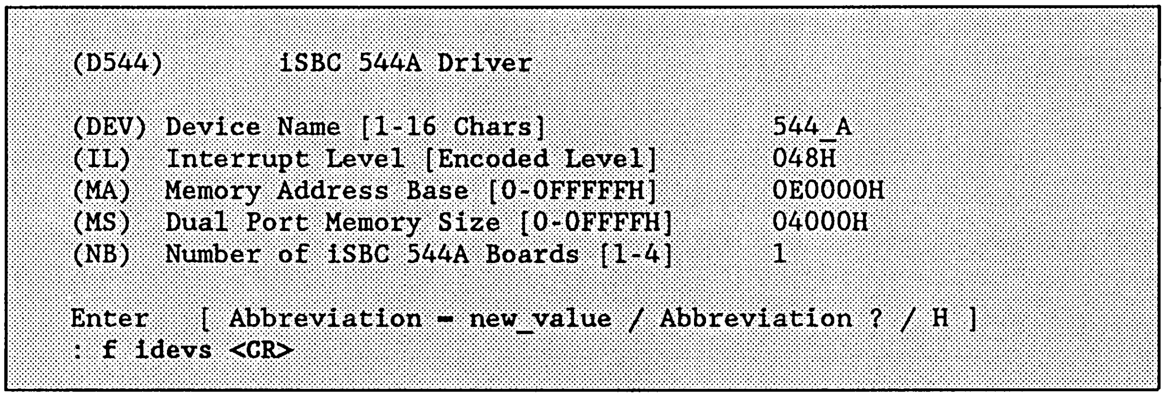
Next, you must change the interrupt level of the iSBC 544A Driver to match the hardware configuration of the target system. To display the "iSBC 544A Driver" screen, enter "T44=y <CR>" as shown in Figure B-22. Although the iSBC 544A Driver is already included in the system definition, setting T44 to "YES" also causes the "iSBC 544A Driver" screen to be displayed.



```
(D544)        iSBC 544A Driver

(DEV) Device Name [1-16 Chars]               544_A
(IL)  Interrupt Level [Encoded Level]        038H
(MA)  Memory Address Base [0-0FFFFFH]        0E0000H
(MS)  Dual Port Memory Size [0-0FFFFH]       04000H
(NB)  Number of iSBC 544A Boards [1-4]       1

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: il=48h <CR>
: <CR>
```

Figure B-23. iSBC® 544A Driver Screen

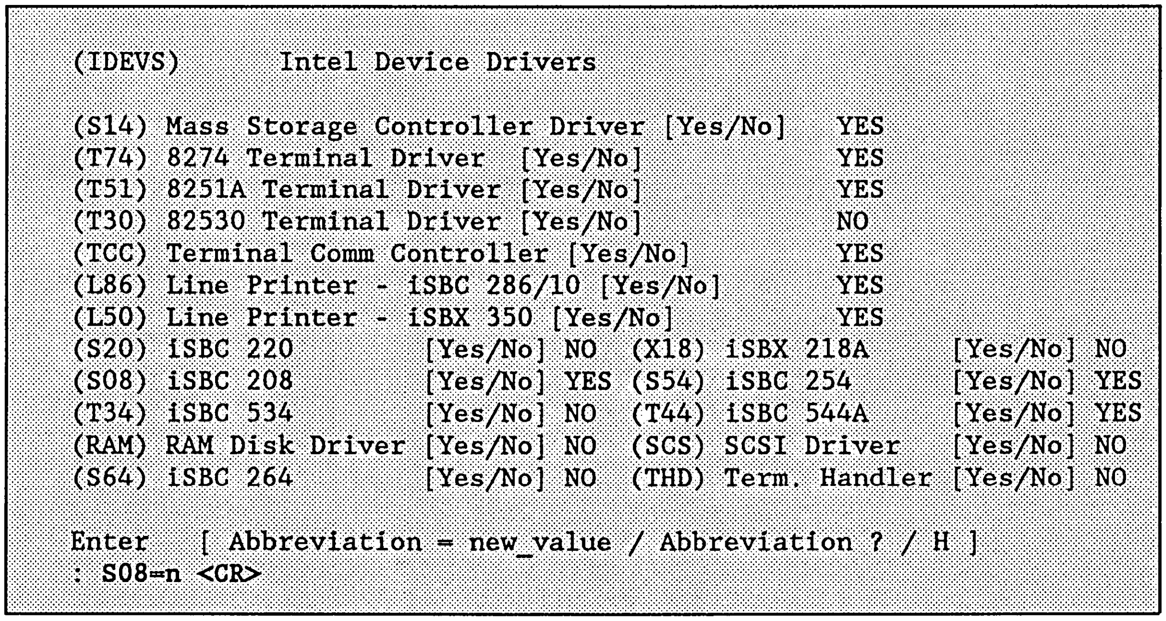
To change the interrupt level for the iSBC 544A Driver, enter "il=48h" as shown in Figure B-23. Then, redisplay the screen to check your entry by entering a carriage return (<CR>).



```
(D544)       iSBC 544A Driver

(DEV) Device Name [1-16 Chars]                544_A
(IL)  Interrupt Level [Encoded Level]         048H
(MA)  Memory Address Base [0-0FFFFFH]         0E0000H
(MS)  Dual Port Memory Size [0-0FFFFH]        04000H
(NB)  Number of iSBC 544A Boards [1-4]        1

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: f idevs <CR>
```

Figure B-24. Completed iSBC® 544A Driver Screen

Again, return to the "Intel Device Drivers" screen by entering "f idevs <CR>" as shown in Figure B-24. This produces the screen in Figure B-25.



```
(IDEVS)      Intel Device Drivers

(S14) Mass Storage Controller Driver [Yes/No]   YES
(T74) 8274 Terminal Driver  [Yes/No]            YES
(T51) 8251A Terminal Driver [Yes/No]            YES
(T30) 82530 Terminal Driver [Yes/No]            NO
(TCC) Terminal Comm Controller [Yes/No]         YES
(L86) Line Printer - iSBC 286/10 [Yes/No]       YES
(L50) Line Printer - iSBX 350 [Yes/No]          YES
(S20) iSBC 220        [Yes/No] NO  (X18) iSBX 218A     [Yes/No] NO
(S08) iSBC 208        [Yes/No] YES (S54) iSBC 254      [Yes/No] YES
(T34) iSBC 534        [Yes/No] NO  (T44) iSBC 544A     [Yes/No] YES
(RAM) RAM Disk Driver [Yes/No] NO  (SCS) SCSI Driver   [Yes/No] NO
(S64) iSBC 264        [Yes/No] NO  (THD) Term. Handler [Yes/No] NO

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: S08=n <CR>
```

Figure B-25. Intel Device Drivers Screen

The final step in configuring the device drivers for the target system is to remove the iSBC 208 Driver. To begin removing the iSBC 208 Driver, enter "S08=n <CR>" as shown in Figure B-25. This causes the query screen in Figure B-26 to appear.

```
First delete iSBC 208 Dinfo screens ! Do you want to delete ??
y <CR>
```

Figure B-26. Query Screen for Deleting iSBC® 208 Driver Screens

To tell the ICU you want to delete the iSBC 208 Driver screens, enter "y <CR>" as shown in Figure B-26. This causes the first "iSBC 208 Driver" screen to appear.

```
(D208)       iSBC 208 Driver

(DEV) Device Name [1-16 Chars]                  208
(IL)  Interrupt Level [Encoded Level]           048H
(ITP) Interrupt Task Priority [0-255]           130
(PA)  Port Address [0-0FFFFH]                   0180H
(MDV) Motor Delay Value [0-0FFFFH]              0H
(BBA) Boundary Buffer Address [0-0FFFFFH]       01600H

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: ^d <CR>
```

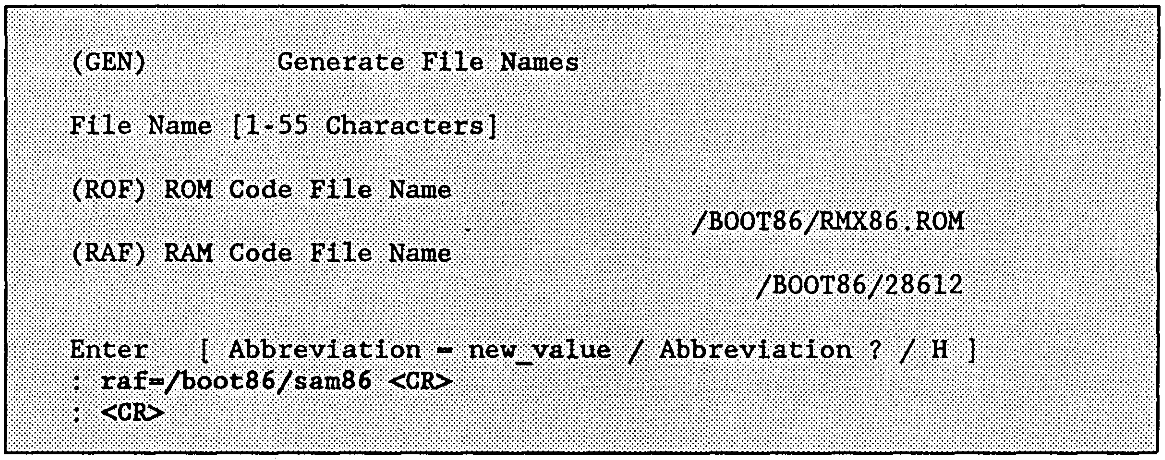
Figure B-27. iSBC® 208 Driver Screen

To delete the iSBC 208 Driver screens, enter "^d <CR>" as shown in Figure B-27. This deletes all iSBC 208 screens and causes the query screen in Figure B-28 to appear.

```
Do you want any/more iSBC 208 DEVICEs ?
f gen <CR>
```

Figure B-28. Query Screen for iSBC® 208 Devices

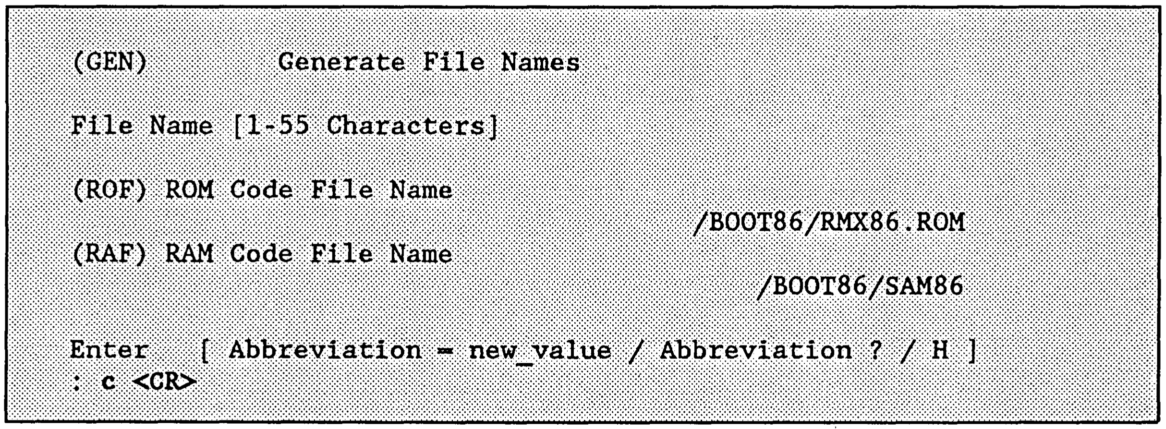
Now that the device drivers for the 80286-based multi-user system have been changed to match the target system, you are ready to generate the system. To produce the "Generate File Names" screen, type "f gen <CR>" as shown in Figure B-28.



```
(GEN)        Generate File Names

File Name [1-55 Characters]

(ROF) ROM Code File Name
                                         /BOOT86/RMX86.ROM
(RAF) RAM Code File Name
                                              /BOOT86/28612

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: raf=/boot86/sam86 <CR>
: <CR>
```

Figure B-29. Generate File Names Screen

On the "Generate File Names" screen, shown in Figure B-29, you must specify the pathname of the new bootloadable file you will be generating. For this application, the file is SAM86, and it will be a RAM code file name. Enter "raf=/boot86/sam86 <CR>". To produce the changed "Generate File Names" screen (shown in Figure B-30), enter a carriage return (<CR>).



```
(GEN)        Generate File Names

File Name [1-55 Characters]

(ROF) ROM Code File Name
                                         /BOOT86/RMX86.ROM
(RAF) RAM Code File Name
                                              /BOOT86/SAM86

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
: c <CR>
```

Figure B-30. New Generate File Names Screen

You are now ready to start the generation phase of the ICU. To do this you must first return to the ICU's main menu screen. Enter "c <CR>" on the screen shown in Figure B-30. This displays the ICU main menu screen shown in Figure B-31.

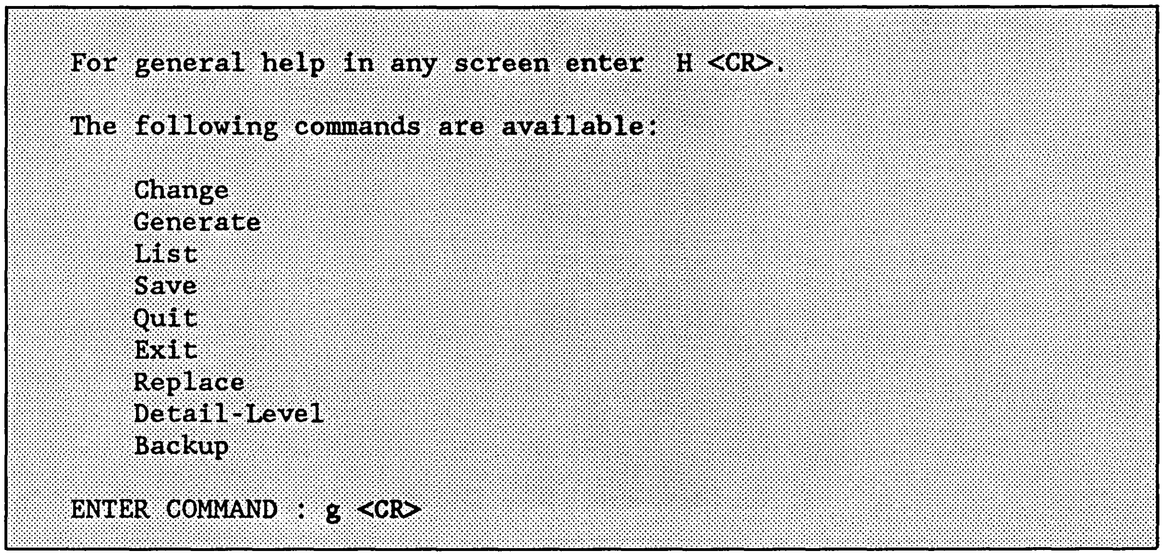

```
For general help in any screen enter  H <CR>.

The following commands are available:

    Change
    Generate
    List
    Save
    Quit
    Exit
    Replace
    Detail-Level
    Backup

ENTER COMMAND : g <CR>
```

Figure B-31. ICU Main Menu Screen

To begin the generation process, enter "g <CR>", as shown in Figure B-31. The ICU asks you to enter a letter to be used as a prefix for the files created during generation. For this example, enter "z <CR>". The ICU displays a message as each subsystem is generated, as shown in Figure B-32.

```
  ENTER a letter to be used as prefix : z <CR>

  The prefix letter is :   Z
  Beginning Nucleus File Generation
  ...................................................DONE
  Beginning Basic I/O System File Generation
  ...................................................DONE
  Beginning Extended I/O System File Generation
  ...................................................DONE
  Beginning Application Loader File Generation
  ...................................................DONE
  Beginning Human Interface File Generation
  ...................................................DONE
  Beginning UDI File Generation
  ...................................................DONE
  Beginning System Debugger File Generation
  ...................................................DONE
  Beginning Submit File Generation
  ...................................................DONE

 To create your application system, type:
    submit SAM86.CSD


 For general help in any screen enter  H <CR>.

 The following commands are available:

    Change
    Generate
    List
    Save
    Quit
    Exit
    Replace
    Detail-Level
    Backup

 ENTER COMMAND : e <CR>

 The Definition File has been written to file:     SAM86.DEF
```

Figure B-32. Generation Phase ICU Screen

When the generation process is completed, the ICU displays the name of the resulting SUBMIT file. In this case, the SUBMIT file is SAM86.CSD. The ICU then continues automatically to the ICU main menu screen where you should enter "e <CR>" to exit the ICU and save the definition file. Upon receiving the Exit command, the ICU informs you that the definition file has been written.

You are now ready to invoke the SUBMIT file SAM86.CSD. You do this by invoking the SUBMIT file in one of the following ways:

```
- SUBMIT sam86 <CR>
```

or

```
- SUBMIT sam86 TO sam86.log <CR>
```

The first method of invoking SAM86.CSD sends the output from the SUBMIT file to your screen. This lets you observe the progress of the SUBMIT file's execution, but does not save the output for later examination. The second method sends the output from the SUBMIT file to a file called SAM86.LOG instead of the screen.

The SUBMIT file assembles all the configuration files generated by the ICU and links the object files with all the libraries required by the subsystems. It then locates the system. Figure B-33 shows a listing of the output from the SUBMIT file. You may notice warning messages. The warning messages are normal and can be ignored. Only error messages must be heeded.

```
- ;
- ;    Nucleus
- ;
- :LANG:ASM86 ZNTABL.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:ASM86 ZNDEVC.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:LINK86 &
**/RMX86/NUCLEUS/NUC1.LIB(NBEGIN), &
**ZNTABL.OBJ, &
**ZNDEVC.OBJ, &
**/RMX86/NUCLEUS/NUC1.LIB, &
**/RMX86/NUCLEUS/NUC2.LIB, &
**/RMX86/NUCLEUS/NUC3.LIB, &
**/RMX86/NUCLEUS/NUC4.LIB, &
**/RMX86/NUCLEUS/NURSLV.LIB &
**TO ZNUCLS.LNK NOPUBLICS
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
WARNING 28:  POSSIBLE OVERLAP
  FILE:  /RMX86/NUCLEUS/NUC2.LIB
  MODULE:  NMINIT
  SEGMENT:  UNNAMED
  CLASS:  UNNAMED
WARNING 28:  POSSIBLE OVERLAP
  FILE:  /RMX86/NUCLEUS/NUC2.LIB
  MODULE:  NTINIT
  SEGMENT:  UNNAMED
  CLASS:  UNNAMED
WARNING 28:  POSSIBLE OVERLAP
  FILE:  /RMX86/NUCLEUS/NUC4.LIB
  MODULE:  NSETIN
  SEGMENT:  UNNAMED
  CLASS:  UNNAMED
```

Figure B-33. Output of Submit File for SAM86.CSD

```
WARNING 28:  POSSIBLE OVERLAP
  FILE:  /RMX86/NUCLEUS/NUC4.LIB
  MODULE:  NSETOS
  SEGMENT:  UNNAMED
  CLASS:  UNNAMED
WARNING 28:  POSSIBLE OVERLAP
  FILE:  /RMX86/NUCLEUS/NURSLV.LIB
  MODULE:  NRSETB
  SEGMENT:  UNNAMED
  CLASS:  UNNAMED
00C1: E$WARNING_EXIT, during submit execution
- ;
- ;    BIOS
- ;
- :LANG:ASM86 ZITABL.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:ASM86 ZICDEV.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:ASM86 ZITDEV.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:LINK86 &
**/RMX86/IOS/IOS.LIB(IBEGIN), &
**ZITABL.OBJ, &
**/RMX86/IOS/IOOPT1.LIB, &
**/RMX86/IOS/IOS.LIB, &
**/RMX86/LIB/RPIFL.LIB &
**TO ZIOS1.LNK NOPUBLICS EXCEPT (rqaiosinittask, rqaiosiniterror)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
WARNING 12: UNRESOLVED SYMBOLS
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

```
- :LANG:LINK86 &
**ZITDEV.OBJ, &
**/RMX86/IOS/XCMDRV.LIB(XTSIF), &
**/RMX86/IOS/XCMDRV.LIB(XTSIO), &
**/RMX86/IOS/XCMDRV.LIB, &
**/RMX86/IOS/X86TSC.LIB, &
**:LANG:PLM86.LIB, &
**/RMX86/LIB/RPIFL.LIB &
**TO ZTSC.LNK NOPUBLICS EXCEPT( TSCINITIO, &
**  TSCFINISHIO, &
**  DINFO_04H, &
**  UINFO_8274, &
**  DINFO_03H, &
**  UINFO_544, &
**  DINFO_0AH, &
**  UINFO_8251, &
**  DINFO_06H, &
**  UINFO_546, &
**  UINFO_546UT, &
**  DINFO_07H, &
**  UINFO_547, &
**  DINFO_08H, &
**  UINFO_18848, &
**  DINFO_09H, &
**  UINFO_18856, &
**  TSCQUEUEIO, &
**  TSCCANCELIO)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
- :LANG:LINK86 &
**ZIOS1.LNK, &
**ZTSC.LNK, &
**ZICDEV.OBJ, &
**/RMX86/IOS/XCMDRV.LIB, &
**/RMX86/IOS/X86DRV.LIB, &
**:LANG:PLM86.LIB, &
**/RMX86/LIB/RPIFC.LIB &
**TO ZIOS.LNK NOPUBLICS EXCEPT(rqaiosinittask, rqaiosiniterror)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

```
- ;
- ;    EIOS
- ;
- :LANG:ASM86 ZETABL.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:ASM86 ZEDEVC.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:ASM86 ZEJOBC.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:LINK86 &
**/RMX86/EIOS/EIOS.LIB(EBEGIN), &
**ZETABL.OBJ, &
**ZEDEVC.OBJ, &
**ZEJOBC.OBJ, &
**/RMX86/EIOS/EIOS.LIB, &
**/RMX86/LIB/EPIFC.LIB, &
**/RMX86/LIB/IPIFC.LIB, &
**/RMX86/LIB/RPIFC.LIB &
**TO ZEIOS.LNK NOPUBLICS EXCEPT(rqeiosinittask,rqeiosiniterror)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
- ;
- ;  Application loader
- ;
- :LANG:PLM86 ZLCONF.P86 COMPACT OPTIMIZE(3) ROM NOTYPE

iRMX I PL/M-86 COMPILER V3.1
Copyright Intel Corporation 1980, 1985, 1987
PL/M-86 COMPILATION COMPLETE.      0 WARNINGS,     0 ERRORS
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

```
- :LANG:LINK86 &
**/RMX86/LOADER/LOADRO.LIB(LBEGIN), &
**ZLCONF.OBJ, &
**/RMX86/LOADER/LOADRO.LIB(LDRENT), &
**/RMX86/LOADER/LJBCFS.LIB, &
**/RMX86/LOADER/LOADRO.LIB, &
**/RMX86/LIB/EPIFC.LIB, &
**/RMX86/LIB/IPIFC.LIB, &
**/RMX86/LIB/RPIFC.LIB &
**TO ZLOADR.LNK NOPUBLICS EXCEPT (rqloaderinittask, rqloaderiniterror)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
- ;
- ;  Human Interface
- ;
- :LANG:PLM86 ZHCONF.P86 COMPACT ROM NOTYPE

iRMX I PL/M-86 COMPILER V3.1
Copyright Intel Corporation 1980, 1985, 1987
PL/M-86 COMPILATION COMPLETE.      1 WARNING,     0 ERRORS

00C1: E$WARNING_EXIT, during submit execution
- :LANG:LINK86 &
**/RMX86/HI/HCLI.LIB(HCLI), &
**/RMX86/HI/HCLI.LIB,       &
**:LANG:PLM86.LIB,      &
**/RMX86/LIB/HPIFL.LIB,      &
**/RMX86/LIB/LPIFL.LIB,      &
**/RMX86/LIB/EPIFL.LIB,      &
**/RMX86/LIB/IPIFL.LIB,      &
**/RMX86/LIB/RPIFL.LIB       &
**TO ZHCLI.LNK &
**SEGSIZE(STACK(02400H)) NOPUBLICS EXCEPT(hcliinit)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
- ;
- ;    CLI
- ;
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

```
- :LANG:LINK86 &
**/RMX86/HI/HI.LIB(HBEGIN), &
**ZHCONF.OBJ, &
**/RMX86/HI/HI.LIB, &
**/RMX86/HI/HUTIL.LIB, &
**ZHCLI.LNK, &
**/RMX86/LIB/HPIFC.LIB, &
**/RMX86/LIB/LPIFC.LIB, &
**/RMX86/LIB/EPIFC.LIB, &
**/RMX86/LIB/IPIFC.LIB, &
**/RMX86/LIB/RPIFC.LIB, &
**:LANG:PLM86.LIB  &
**TO ZHI.LNK &
**SEGSIZE(STACK(0)) NOPUBLICS EXCEPT(rqhiinittask,rqhiiniterror)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
WARNING 14:  GROUP ENLARGED
  FILE:  ZHCLI.LNK
  GROUP:  CGROUP
  MODULE:  HCLI
00C1: E$WARNING_EXIT, during submit execution
- ;
- ;   UDI
- ;
- :LANG:ASM86 ZUDICN.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:LINK86 &
**/RMX86/UDI/UDI.LIB(UBEGIN), &
**ZUDICN.OBJ, &
**/RMX86/UDI/UDI.LIB, &
**/RMX86/LIB/HPIFC.LIB, &
**/RMX86/LIB/LPIFC.LIB, &
**/RMX86/LIB/EPIFC.LIB, &
**/RMX86/LIB/IPIFC.LIB, &
**/RMX86/LIB/RPIFC.LIB, &
**:LANG:PLM86.LIB &
**TO ZUDI.LNK NOPUBLICS EXCEPT(rqudiinittask,rqudiiniterror)
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

```
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
- ;
- ;   SDB
- ;
- :LANG:ASM86 ZSDBCN.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:LINK86 &
**/RMX86/SDB/SDB.LIB(SBEGIN), &
**ZSDBCN.OBJ, &
**/RMX86/SDB/SDB.LIB, &
**/RMX86/LIB/RPIFC.LIB &
**TO ZSDB.LNK NOPUBLICS EXCEPT(rqsdbinittask, rqsdbiniterror)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
- :ICU:icu86.862 SAM86.def Z
iRMX I Interactive Configurator For iRMX I Stage 2, V8.0
Copyright 1982, 1989 Intel Corporation - All Rights Reserved
- SUBMIT ICULOC.CSD
- :LANG:LOC86 &
**ZUDI.LNK TO ZUDI &
** SEGSIZE(STACK(0)) PRINT(ZUDI.MP2) &
**ADDRESSES(CLASSES(CODE(042480H),DATA(045540H))) NOINITCODE OC(PURGE)
iRMX I 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation

WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  STACK
WARNING 66:  START ADDRESS NOT SPECIFIED IN OUTPUT MODULE
00C1: E$WARNING_EXIT, during submit execution
- :LANG:LOC86 &
**ZSDB.LNK TO ZSDB &
** SEGSIZE(STACK(0)) PRINT(ZSDB.MP2) &
**ADDRESSES(CLASSES(CODE(036BF0H),DATA(045170H))) NOINITCODE OC(PURGE)
iRMX I 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

```
WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  STACK
WARNING 66:  START ADDRESS NOT SPECIFIED IN OUTPUT MODULE
00C1: E$WARNING_EXIT, during submit execution
- :LANG:LOC86 &
**ZHI.LNK TO ZHI &
** SEGSIZE(STACK(0)) PRINT(ZHI.MP2) &
**ADDRESSES(CLASSES(CODE(01A8F0H),DATA(044280H))) NOINITCODE OC(PURGE)
iRMX I 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation

WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  STACK
WARNING 18:  SIZE OF GROUP EXCEEDS 64K
  GROUP:  CGROUP
00C1: E$WARNING_EXIT, during submit execution
- :LANG:LOC86 &
**ZLOADR.LNK TO ZLOADR &
** SEGSIZE(DATA(2),STACK(0)) PRINT(ZLOADR.MP2) &
**ADDRESSES(CLASSES(CODE(03FE40H),DATA(045570H))) NOINITCODE OC(PURGE)
iRMX I 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation

WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  STACK
WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  DATA
WARNING 66:  START ADDRESS NOT SPECIFIED IN OUTPUT MODULE
00C1: E$WARNING_EXIT, during submit execution
- :LANG:LOC86 &
**ZEIOS.LNK TO ZEIOS &
** SEGSIZE(STACK(0)) PRINT(ZEIOS.MP2) &
**ADDRESSES(CLASSES(CODE(03C0F0H),DATA(0455B0H))) NOINITCODE OC(PURGE)
iRMX I 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation

WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  STACK
WARNING 66:  START ADDRESS NOT SPECIFIED IN OUTPUT MODULE
00C1: E$WARNING_EXIT, during submit execution
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

```
- :LANG:LOC86 &
**ZIOS.LNK TO ZIOS &
** SEGSIZE(STACK(0)) PRINT(ZIOS.MP2) &
**ADDRESSES(CLASSES(CODE(001200H),DATA(045480H))) NOINITCODE OC(PURGE)
iRMX I 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation

WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  STACK
WARNING 66:  START ADDRESS NOT SPECIFIED IN OUTPUT MODULE
00C1: E$WARNING_EXIT, during submit execution
- :LANG:LOC86 &
**ZNUCLS.LNK TO ZNUCLS &
** SEGSIZE(DATA(2),STACK(0)) PRINT(ZNUCLS.MP2) &
**ADDRESSES(CLASSES(CODE(030390H),DATA(045590H))) NOINITCODE OC(PURGE)
iRMX I 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation

WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  STACK
WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  DATA
WARNING 66:  START ADDRESS NOT SPECIFIED IN OUTPUT MODULE
00C1: E$WARNING_EXIT, during submit execution
- END SUBMIT ICULOC.CSD
- SUBMIT ICUROT.CSD
- :LANG:ASM86 ROOT.A86
iRMX I 8086/87/88/186 MACRO ASSEMBLER, V2.1
Copyright 1980, 1981, 1982 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND
- :LANG:LINK86 &
**/RMX86/NUCLEUS/CROOT.LIB(RBEGIN), &
**ROOT.OBJ, &
**/RMX86/NUCLEUS/CROOT.LIB &
** TO CROOT.LNK NOPUBLICS EXCEPT(SAB_LIST_PTR,NUMBER_SABS, &
**RQSTARTADDRESS,ROOTTASKSTATUS)
iRMX I 8086 LINKER, V2.7
Copyright 1984 Intel Corporation
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

```
- :LANG:LOC86 &
**CROOT.LNK TO ROOT &
** SEGSIZE(STACK(0)) ORDER(CLASSES(DATA,STACK)) PRINT(ROOT.MP2) &
**ADDRESSES(CLASSES(CODE(0455C0H),DATA(046310H))) INITCODE(0455C0H) &
**OC(NOLI, NOCM, NOSB) PC(NOLI, PL, NOCM, NOSB)
iRMX I 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation

WARNING 63:  SS AND SP REGISTERS NOT INITIALIZED
WARNING 64:  DS REGISTER NOT INITIALIZED
WARNING 26:  DECREASING SIZE OF SEGMENT
  SEGMENT:  STACK
00C1: E$WARNING_EXIT, during submit execution
- END SUBMIT ICUROT.CSD
- DELETE /BOOT86/SAM86
/BOOT86/SAM86, deleted
- :LANG:LIB86
iRMX I 8086 LIBRARIAN V2.1
*CREATE /BOOT86/SAM86
*ADD ZSDB TO /BOOT86/SAM86
*ADD ZNUCLS TO /BOOT86/SAM86
*ADD ZIOS TO /BOOT86/SAM86
*ADD ZEIOS TO /BOOT86/SAM86
*ADD ZLOADR TO /BOOT86/SAM86
*ADD ZHI TO /BOOT86/SAM86
*ADD ZUDI TO /BOOT86/SAM86
*ADD ROOT TO /BOOT86/SAM86
*EXIT
- END SUBMIT sam86.csd
```

Figure B-33. Output of Submit File for SAM86.CSD (continued)

This ends the output from the SUBMIT file SAM86.CSD. A bootable file named /BOOT86/SAM86 contains the entire example system. You are now ready to bootload your new executable system.

B.6 BOOTING THE EXAMPLE SYSTEM

This section explains how to boot the example system from the iSDM monitor.

To boot the example system, you must shutdown the system properly by invoking the SHUTDOWN command. To do this, invoke the SUPER command and enter the password 'passme' when prompted. This causes you to become the system manager without logging on as the user "super". Type:

```
- SUPER <CR>
```

Respond with the password 'passme' to the password prompt. Then type:

```
- SH <CR>
```

The command "SH" is the alias for the Human Interface command invocation ":SYSTEM:SHUTDOWN W=0". After the SHUTDOWN command displays its shutdown complete message, reset the system by pressing the RESET button or turning the RESET switch on the front panel of your microcomputer.

The System Confidence Test (SCT) will start executing in a few seconds. Enter an uppercase U in response to the x's being printed on the screen. When your system console displays the monitor prompt, enter the following monitor command to bootstrap load the newly created version of the operating system:

```
.b /boot86/sam86 <CR>
```

Once the operating system bootstrap loads and signs on, it indicates you have successfully generated a version of the operating system. The logon banner and prompt will be displayed. Now log back on to the system using the name "world" and a carriage-return for the password.

PROGRAMMING AN iRMX® I SYSTEM INTO PROM DEVICES C

C.1 INTRODUCTION

This appendix provides an example of the procedures used to place the iRMX I Operating System into ROM of a 286-based system. All software generation described assumes you are using an Intel System 300 Series Microcomputer.

The example places the iSDM monitor and the iRMX Bootstrap Loader in ROM along with the generated operating system. A system such as this executes the iSDM monitor code during the initial power-up sequence. This example includes the iSDM monitor and iRMX Bootstrap Loader for several reasons. First, while you develop the ROM-based system, you can bootstrap load RAM-based versions of the operating system rather than having to switch PROM devices if you are using one processor board. Second, the iSDM monitor, while not allowing breakpoints in PROM devices, does allow you to disassemble and examine memory in both ROM and RAM. You can use this feature to determine the correct code is in the correct locations.

C.2 REQUIREMENTS

To use the procedures outlined in this appendix, you must have the following hardware and software:

- A system defined during configuration as residing in ROM.
- The iRMX I.8 Operating System.
- The iPPS software for the iRMX I.8 Operating System.
- A 286-based system.
- An iUP-200/201 Universal Programmer with a 'FAST 27/K' module with 27512 support.
- The iSDM monitor, which must be purchased separately from the iRMX I Operating System.
- Four 27512 EPROM devices, labeled U2, U3, U5, and U6.

C.3 CONFIGURING A ROM-BASED SYSTEM

Before you can program your system into PROM devices, you must modify a number of parameters in your definition file. These examples assume that you start with the Intel-supplied definition file 28612.DEF located in directory /RMX86/ICU.

To begin the example, you should make a copy of the definition file in a separate directory. To do this, create a new directory in which to do the system generation. The definition file should have a .DEF extension, the boot-loadable system should have a .86 extension, and the system to be loaded into ROM should have a .ROM extension. Entering the three following commands creates a new directory called ROMSYS, attaches you to that directory, and invokes the ICU placing a copy of the definition file 28612.DEF into the new directory. (This example assumes your :HOME: directory is your current default directory.)

```
- CREATEDIR romsys  <CR>
- ATTACHFILE romsys  <CR>
- ICU86 /rmx86/icu/28612.def TO romsys.def  <CR>
```

As part of the configuration process, you must perform the following three things in order to fit the iRMX I system developed in these examples into ROM:

- Delete the System Debugger from the system.
- Replace the iRMX I.8 CLI with the iRMX 86 R7.0 CLI.
- Delete the iSBC 254 and iSBC 544A device drivers.

Having invoked the ICU, you now can begin to make the necessary configuration changes. Start by modifying the memory screens to define the RAM and ROM memory locations the system requires. Do this by first selecting the "RAM Memory" screen with the following command:

```
c ramem  <CR>
```

Entering the previous command causes the "RAM Memory" screen to appear as follows:

```
(RAMEM)    RAM Memory

        Memory =       low    ,    high
                   [0-0FFFFH]  [0-0FFFFH]

[ 1]    Memory =      0120H   ,    0DFFFH
[ 2]    Memory =

Enter Changes [ Number= new_value / ^D Number / ? / H ]
:
```

Before entering the new memory addresses, you must delete line 1 by entering the following command:

```
^d 1  <CR>
```

After the screen reappears, enter the low and high addresses of reserved RAM as follows:

```
1 = 0120h,bfffh  <CR>
<CR>
```

After entering the new memory locations, the "RAM Memory" screen appears as follows:

```
(RAMEM)    RAM Memory

        Memory =       low    ,    high
                   [0-0FFFFH]  [0-0FFFFH]

[ 1]    Memory =      0120H   ,    0BFFFH
[ 2]    Memory =

Enter Changes [ Number= new_value / ^D Number / ? / H ]
:
```

Now, display the "ROM Memory" screen by entering "<CR>", as shown in the previous screen. You must also adjust this memory screen as a ROM-based system uses different memory locations than the RAM-based system defined in the definition file. Change the "ROM Memory" screen to contain the memory locations C000H to 0F7FFH. The following screen shows the "ROM" screen after making the changes.

```
(ROMEM)   ROM Memory

          Memory =        low    ,    high
                      [0-0FFFFH]  [0-0FFFFH]

[ 1]      Memory =      0C000H ,    0F7FFH
[ 2]      Memory =

Enter Changes [ Number= new_value / ^D Number / ? / H ]
:
```

After changing the memory screens, request the "Sub-systems" screen by entering the following command:

```
f sub  <CR>
```

```
(SUB) Sub-systems

(UDI) Universal Development Interface [Yes/No]   YES
(HI)  Human Interface [Yes/No]                   REQ
(AL)  Application Loader [Yes/No]                REQ
(RFA) Remote File Access [Yes/No]                NO
(EIO) Extended I/O System [Yes/No]               REQ
(BIO) Basic I/O System [Yes/No]                  REQ
(SDB) System Debugger [Yes/No]                   YES
(DDB) Dynamic Debugger [Yes/No]                  NO
(THD) Terminal Handler [Yes/No]                  NO

Enter    [ Abbreviation = new_value / Abbreviation ? / H ]
:
```

Here, you must delete the System Debugger because of size considerations. Delete the SDB by entering the following command:

```
sdb=no  <CR>
```

Next, request the "Human Interface" screen by entering the following command:

```
f hi  <CR>
```

```
(HI)         Human Interface

(ICL) Initial Command Line Size [0-65535]                        256
(CNM) Command Name Length [1-255]                                64
(SYS) System Directory [1-45 Chars]
                                                               :SD:SYSTEM
(RIP) Resident Initial Program [IntelCLI/1-45 Chars]
                                                                 INTELCLI
(UXC) User Extension for Intel CLI [1-45 Chars]

(SS)  Initial Program Stack Size [0-0FFFFH]                      02400H
(PMI) Human Interface Pool Minimum [0-0FFFFH]                    0260H
(PMA) Human Interface Pool Maximum [0-0FFFFH]                    0FFFFH
(DTN) Default Terminal Name [1-6 Chars]                          VT100
(RU)  Resident User [Yes/Recovery/None]                          NONE

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
:
```

Change the resident initial program (CLI) by entering the following command:

```
rip=/rmx86/hi/r7cli.lnk  <CR>
```

The iRMX I.8 CLI is considerably larger than the iRMX 86 R7.0 CLI and cannot be used in this example due to size restrictions.

Next, request the "Intel Device Driver" screen by entering the following command:

```
f idevs  <CR>
```

```
(IDEVS)      Intel Device Drivers

(S14) Mass Storage Controller Driver [Yes/No]    YES
(T74) 8274 Terminal Driver  [Yes/No]             YES
(T51) 8251A Terminal Driver [Yes/No]             NO
(T30) 82530 Terminal Driver [Yes/No]             NO
(TCC) Terminal Comm Controller [Yes/No]          YES
(L86) Line Printer - iSBC 286/10 [Yes/No]        YES
(L50) Line Printer - iSBX 350 [Yes/No]           NO
(S20) iSBC 220          [Yes/No] NO   (X18) iSBX 218A     [Yes/No] NO
(S08) iSBC 208          [Yes/No] YES  (S54) iSBC 254      [Yes/No] YES
(T34) iSBC 534          [Yes/No] NO   (T44) iSBC 544A     [Yes/No] YES
(RAM) RAM Disk Driver [Yes/No] NO     (SCS) SCSI Driver   [Yes/No] NO
(S64) iSBC 264          [Yes/No] NO   (THD) Term. Handler [Yes/No] NO

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
:
```

Delete the iSBC 254 and iSBC 544A device drivers by entering the following commands:

```
s54=n  <CR>
First delete iSBC 254 Dinfo screens ! Do you want to delete ??
y  <CR>
^d  <CR>
Do you want any/more iSBC 254 DEVICEs ?  f idevs  <CR>
t44=n  <CR>
First delete 544A  Terminal Dinfo screens! Do you want to delete ??
y  <CR>
^d  <CR>
Do you want any/more iSBC 544A DEVICEs ?
n  <CR>
```

Next you must include the various subsystems of the operating system on the "ROM Code" screen. Request the "ROM Code" screen by entering:

```
f rom  <CR>
```

```
(ROM)        ROM Code

(UIR) UDI in ROM [Yes/No]                          NO
(HIR) Human Interface in ROM [Yes/No]              NO
(ALR) Application Loader in ROM [Yes/No]           NO
(EIR) Extended I/O System in ROM [Yes/No]          NO
(BIR) Basic I/O System in ROM [Yes/No]             NO
(THR) Terminal Handler in ROM [Yes/No]             NO
(SIR) SDB in ROM [Yes/No]                          NO
(NIR) Nucleus in ROM [Yes/No]                      NO
(RIR) Root Job in ROM [Yes/No]                     NO

Enter   [ Abbreviation = new_value / Abbreviation ? / H ]
:
```

To include the operating system subsystems in ROM, enter the following commands:

```
uir=yes  <CR>
hir=yes  <CR>
alr=yes  <CR>
eir=yes  <CR>
bir=yes  <CR>
nir=yes  <CR>
rir=yes  <CR>
```

Now, request the "Generate File Names" screen by entering the following command:

```
f gen  <CR>
```

The "Generate File Names" screen is where you define the pathname of the file containing the ROM-based system.

```
(GEN)        Generate File Names

File Name [1-55 Characters]

(ROF) ROM Code File Name
                                                 /BOOT86/RMX86.ROM
(RAF) RAM Code File Name
                                                     /BOOT86/28612

Enter    [ Abbreviation = new_value / Abbreviation ? / H ]
:
```

Enter the pathname :$:ROMSYS.ROM as shown below:

```
rof=:$:romsys.rom  <CR>
c  <CR>
```

Entering "c <CR>" as shown above displays the ICU main menu screen.

C.4 GENERATING/BUILDING THE SYSTEM

You are now ready to generate the definition file and build the system. The last step of the previous section caused the ICU main screen to appear. From this screen, enter the Generate (G) command to generate files.

```
For general help in any screen enter  H <CR>.

The following commands are available:

    Change
    Generate
    List
    Save
    Quit
    Exit
    Replace
    Detail-Level
    Backup

ENTER COMMAND : g <CR>
```

After entering "g <CR>" with no prefix, the ICU informs you as each subsystem is generated (see Figure B-32 for an example). When the system has been generated, the ICU returns to the main menu screen.

Enter the Exit (E) command to write the definition file and exit the ICU as follows:

```
E <CR>
```

After entering this command, the ICU informs you that the definition file has been written. It issues the following message before returning control to the command line:

```
The Definition File has been written to file:  ROMSYS.DEF
```

You are now ready to invoke the SUBMIT file ROMSYS.CSD that builds the system. The ICU created this SUBMIT file as part of the generation process. Execution of the SUBMIT file generates the application system with the pathname :$:ROMSYS.ROM.

Execute ROMSYS.CSD by entering one of the following lines:

```
- SUBMIT romsys.csd <CR>

        or

- SUBMIT romsys.csd TO romsys.log <CR>
```

During execution, ROMSYS.CSD invokes the second stage of the ICU (ICU86.862). The second stage produces two other SUBMIT files called ICULOC.CSD and ICUROT.CSD, which are executed immediately after the ICU's second stage. These two SUBMIT files (shown in Figures C-1 and C-2) contain addresses you will need when preparing to program your system into PROM.

```
:LANG:LOC86 &
UDI.LNK TO /BOOT86/ROMSYS.ROMUDI &
 SEGSIZE(STACK(0)) PRINT(UDI.MP2) &
ADDRESSES(CLASSES(CODE(0F00C0H),DATA(0013D0H))) NOINITCODE OC(PURGE)
:LANG:LOC86 &
HI.LNK TO /BOOT86/ROMSYS.ROMHI &
 SEGSIZE(STACK(0)) PRINT(HI.MP2) &
ADDRESSES(CLASSES(CODE(0D9410H),DATA(001200H))) NOINITCODE OC(PURGE)
:LANG:LOC86 &
LOADR.LNK TO /BOOT86/ROMSYS.ROMLOADR &
 SEGSIZE(DATA(2),STACK(0)) PRINT(LOADR.MP2) &
ADDRESSES(CLASSES(CODE(0EDA80H),DATA(001400H))) NOINITCODE OC(PURGE)
:LANG:LOC86 &
EIOS.LNK TO /BOOT86/ROMSYS.ROMEIOS &
 SEGSIZE(STACK(0)) PRINT(EIOS.MP2) &
ADDRESSES(CLASSES(CODE(0E9D30H),DATA(001440H))) NOINITCODE OC(PURGE)
:LANG:LOC86 &
IOS.LNK TO /BOOT86/ROMSYS.ROMIOS &
 SEGSIZE(STACK(0)) PRINT(IOS.MP2) &
ADDRESSES(CLASSES(CODE(0C0000H),DATA(001310H))) NOINITCODE OC(PURGE)
:LANG:LOC86 &
NUCLS.LNK TO /BOOT86/ROMSYS.ROMNUCLS &
 SEGSIZE(DATA(2),STACK(0)) PRINT(NUCLS.MP2) &
ADDRESSES(CLASSES(CODE(0E34D0H),DATA(001420H))) NOINITCODE OC(PURGE)
```

Figure C-1. ICULOC.CSD File

```
:LANG:ASM86 ROOT.A86
:LANG:LINK86 &
/RMX86/NUCLEUS/CROOT.LIB(RBEGIN), &
ROOT.OBJ, &
/RMX86/NUCLEUS/CROOT.LIB &
 TO CROOT.LNK NOPUBLICS EXCEPT(SAB_LIST_PTR,NUMBER_SABS, &
RQSTARTADDRESS,ROOTTASKSTATUS)
:LANG:LOC86 &
CROOT.LNK TO /BOOT86/ROMSYS.ROMROOT &
 SEGSIZE(STACK(0)) ORDER(CLASSES(DATA,STACK)) PRINT(ROOT.MP2) &
ADDRESSES(CLASSES(CODE(0F1EC0H),DATA(001450H))) INITCODE(0F1EC0H) &
OC(NOLI, NOCM, NOSB) PC(NOLI, PL, NOCM, NOSB)
```

Figure C-2. ICUROT.CSD File

After the SUBMIT file completes execution, create and attach a working directory for IPPS by entering:

```
- CREATEDIR workdir <CR>
- ATTACHFILE workdir AS :f1: <CR>
```

Then use the COPY command to copy the files resulting from the SUBMIT file's execution to the new working directory.

```
- COPY :sd:boot86/romsys.romudi   TO :f1:udi  <CR>
- COPY :sd:boot86/romsys.romhi    TO :f1:hi   <CR>
- COPY :sd:boot86/romsys.romloadr TO :f1:load <CR>
- COPY :sd:boot86/romsys.romeios  TO :f1:eios <CR>
- COPY :sd:boot86/romsys.romios   TO :f1:ios  <CR>
- COPY :sd:boot86/romsys.romnucls TO :f1:nucl <CR>
- COPY :sd:boot86/romsys.romroot  TO :f1:root <CR>
```

Now that the system has been generated, you can generate versions of the iSDM monitor and the Bootstrap Loader to be programmed into the PROM devices.

C.4.1 Including the iSDM™ Monitor and the Bootstrap Loader in the PROM Devices

This section explains how to include the iSDM monitor and iRMX Bootstrap Loader as part of the operating system that resides in ROM. With this type of system, the iSDM monitor code executes when the system is powered up. In order to create a system that includes the iSDM monitor and the Bootstrap Loader you must prepare two files; one file for each piece of software.

C.4.1.1 Generating the iSDM™ Monitor

First you must generate a version of the iSDM monitor for the iSBC 286/10(A) board. To do this, copy the file 28610A.A28 to a new file called 28610A.A86, then modify the new file. On iRMX I systems, 28610A.A28 normally resides in the directory /SDM. Invoke AEDIT on the 28610A.A86 file and make the following changes:

- Change the line:

  ```
  %cpu(80286,8)
      to
  %cpu(8086,8)
  ```

  This tells iSDM to act as if it were running on an 8086 processor.

- Comment out the extended addressing line as follows:

  ```
  ;%extended_addressing(286/10a)
  ```

- Ensure that the following line is not commented out:

  ```
  %bootstrap(0FE40:0,manual)
  ```

  This tells iSDM that the Bootstrap Loader is to be included in the PROM devices along with the monitor.

Finally, generate the iSDM monitor by entering the following:

```
- SUBMIT cnf286(28610a) <CR>
```

The iSDM monitor is always located at address 0F8000H. The iSDM monitor places values into the reset vector and receives control on power-up or reset. Refer to the *iSDM™ System Debug Monitor User's Guide* for more information on generating the iSDM monitor.

C.4.1.2 Generating the Bootstrap Loader

Next, generate a version of the iRMX Bootstrap Loader first stage. Create and attach a new directory, then copy the Bootstrap Loader configuration files to the new directory by entering:

```
- CREATEDIR rom_examp  <CR>
- ATTACHFILE rom_examp  <CR>
- SUBMIT /bsl/setup.csd  <CR>
```

Due to the limited amount of space left over for the Bootstrap Loader, you cannot leave all available devices selected within the first stage configuration file called BS1.A86. Consequently, you must edit BS1.A86 to retain only the MSC device driver. You must also change the CPU type specified in the CPU macro to 80286.

To modify the file, invoke AEDIT on BS1.A86 and make the following changes:

- Change the line:

```
%cpu(80386)
    to
%cpu(80286)
```

- At the end of the file, comment out the following device macro lines by replacing the percent characters (%) at the beginning of the lines with semi-colons (;):

```
%device(af0, 0, deviceinit208gen, deviceread208gen)
%device(af1, 1, deviceinit208gen, deviceread208gen)
%device(s0, 0, deviceinitscsi, devicereadscsi)
%device(sx1410a0, 0, deviceinitscsi, devicereadscsi, sasi_x1410a)
%device(sx1410b0, 0, deviceinitscsi, devicereadscsi, sasi_x1410b)
%device(smf0, 2, deviceinitscsi, devicereadscsi, sasi_x1420mf)
%device(pmf0, 0, deviceinit218A, deviceread218A)
%device(pb0, 0, deviceinit251, deviceread251)
%device(b0, 0, deviceinit254, deviceread254)
%device(ba0, 0, deviceinit264, deviceread264)
%device(r0, 0, deviceinit552A, deviceread552A)
```

  This leaves only the device macro lines that pertain to the MSC device driver.

You must also edit the first stage configuration SUBMIT file (BS1.CSD) to assemble and link in only the MSC file. To do this, invoke AEDIT on BS1.CSD and make these changes:

- Comment out the following ASM86 invocation lines by inserting a semi-colon (;) at the beginning of each line:

```
asm86 b208.a86   macro(50) object(b208.obj)  print(b208.lst)
asm86 b218a.a86  macro(50) object(b218a.obj) print(b218a.lst)
asm86 b251.a86   macro(50) object(b251.obj)  print(b251.lst)
asm86 b254.a86   macro(50) object(b254.obj)  print(b254.lst)
asm86 b264.a86   macro(50) object(b264.obj)  print(b264.lst)
asm86 b552a.a86  macro(50) object(b552a.obj) print(b552a.lst)
asm86 bscsi.a86  macro(50) object(bscsi.obj) print(bscsi.lst)
```

- Comment out the following lines in the LINK86 invocation by inserting an ampersand (&) at the beginning of each line:

```
b208.obj,                &
b218a.obj,               &
b251.obj,                &
b254.obj,                &
b264.obj,                &
b552a.obj,               &
bscsi.obj,               &
```

Now you are ready to generate the Bootstrap Loader. Execute the Bootstrap Loader SUBMIT file by entering the following command:

```
- SUBMIT bsl(OFE400H,0B8000h,BS1) over bsl.out echo  <CR>
```

Complete details on these actions are available in the *iRMX® Bootstrap Loader Reference Manual.*

At this point, the files for the iSDM monitor and the Bootstrap Loader are generated. Copy the resulting files, BS1 and 28610A, to the IPPS working directory you created earlier. The following commands copy the two files:

```
- COPY bsl TO :fl:bsl  <CR>
- COPY /sdm/28610a TO :fl:28610a  <CR>
```

Finally, attach the IPPS working directory as your current directory by entering:

```
- ATTACHFILE :fl:  <CR>
```

The next few sections describe how to program the PROM devices.

C.4.2 Setting Up the iUP 201 PROM Programmer

Perform the following three steps to set up the iUP 201 PROM Programmer:

1. Make sure that the RS-232A line is connected from the iUP 201 programmer to the iRMX I system.
2. Insert the 'FAST 27K' module into the iUP 201 28-pin socket and turn on the power to the iUP 201 Universal Programmer.
3 Press the ONLINE button on the iUP 201 front panel.

WARNING

While following the steps outlined in this section, you must closely adhere to any warnings or cautions given in the *iUP-200/201 Universal Programmer User's Guide.*

C.4.3 Formatting the Operating System PROM Files

Before you can program the operating system into PROM, you must split the PROM image files into even and odd bytes. To do this, first use AEDIT to create a SUBMIT file called PROM86.CSD (shown in Figure C-3). Then create a second SUBMIT file called ROM.CSD (shown in Figure C-4). PROM86.CSD invokes IPPS to split one PROM file into two files: one containing even bytes and one containing odd bytes. ROM.CSD invokes PROM86.CSD on each of the PROM files.

```
;
;  *-*-*  PROM86.CSD  *-*-*
;
;  Format iSDM monitor code and burn into two 27512 EPROMs.
;
;  Invocation: submit prom86(source file,omf type,EPROM start address)
;
;
delete %0.evn, %0.odd
;
ipps


initialize %1

;
; Format monitor code into even and odd addressed bytes.
;
format %0(%2) p
3
2
1
0 to %0.evn
1 to %0.odd

exit
```

Figure C-3. PROM86.CSD File

Notice that Figure C-4 contains start addresses (shown in bold) for each PROM file. Although these addresses are supplied in this example, normally you would have to provide them. The start addresses shown for the iSDM monitor and the Bootstrap Loader are always the same: 0F8000H and 0FE400H. The addresses for the iRMX I subsystems can be found in the ICULOC.CSD file (shown in bold in Figure C-1). The root job starting address can be obtained from the ICUROT.CSD file (shown in bold in Figure C-2).

```
; This file breaks all of the files to be prommed
; into even and odd bytes

submit prom86(28610a,86,0f8000h) over rom.log e
submit prom86(bsl,86,0fe400h) after rom.log e
submit prom86(ios,86,0c0000h) after rom.log e
submit prom86(hi,86,0d9410h) after rom.log e
submit prom86(nucl,86,0e34d0h) after rom.log e
submit prom86(eios,86,0e9d30h) after rom.log e
submit prom86(load,86,0eda80h) after rom.log e
submit prom86(udi,86,0f00c0h) after rom.log e
submit prom86(root,86,0f1ec0h) after rom.log e
```

Figure C-4. ROM.CSD File

Once you have created the required SUBMIT files, enter the following:

```
- SUBMIT rom.csd  <CR>
```

When ROM.CSD ends, the PROM image files have been split into files containing even and odd bytes.

C.4.4 Programming the PROM Devices

Before you can copy the split PROM files into EPROM, you must create the two SUBMIT files shown in Figures C-5 and C-6: COPY1ST.CSD and COPYLAST.CSD. These SUBMIT files invoke IPPS and issue commands that copy the PROM files into EPROM. You will invoke each of these SUBMIT files once for the even byte files and once for the odd byte files.

```
; Copies the first part of the iRMX I system into EPROM
;
ipps
type 27512
init 86
copy ios.%0(0) to p(0)
exit
```

Figure C-5. COPY1ST.CSD File

```
; Copies the last part of the iRMX I system into EPROM
;
ipps
type 27512
init 86
copy 28610a.%0(0) to p(0c000)
copy bsl.%0(0) to p(0f200)
copy nucl.%0(0) to p(1a68)
copy eios.%0(0) to p(4e98)
copy load.%0(0) to p(6d40)
copy udi.%0(0) to p(8060)
copy root.%0(0) to p(8f60)
exit
```

Figure C-6. COPYLAST.CSD File

The bold parameters in Figures C-5 and C-6 are offsets that tell IPPS where to place the code in the PROM devices. Although these offsets are supplied in this example, you would normally calculate them yourself. To calculate the offsets, do the following:

1. Examine the ICULOC.CSD file to find out which subsystem file has the lowest starting address. In this example, the lowest starting address is C0000H for the IOS subsystem. This subsystem will start at offset 0 in PROM. Notice that the offset for the IOS in Figure C-5 is 0.

2. Calculate the offsets for each subsystem by subtracting the lowest starting address (C0000H) from the subsystem's starting address and dividing the result by 2. Only the lower four hex digits of the result are significant. If the result is larger than four hex digits, it indicates that the subsystem must be programmed into the second set of PROM devices.

   For example, the offset for the Nucleus can be calculated as follows:

   (0E34D0H - 0C0000H) / 2 = 11A68H

   Because the result contains five hex digits, the Nucleus will start in the second set of PROM devices. Only the lower four hex digits (1A68H) are significant.

The commands to copy the Human Interface file are not included in either SUBMIT file. This is because the Human Interface does not completely fit in one EPROM and, thus, must be copied by invoking IPPS manually.

To place the generated system into the PROM devices, begin by inserting the first PROM device (U2) into the active socket of the PROM programmer. Then enter the following:

```
- SUBMIT copylst(evn)  <CR>
```

This invokes IPPS and copies the even bytes of the Basic I/O System into the U2 PROM device.

Next, manually invoke IPPS to begin copying the Human Interface into U2. During the copy, the IPPS software automatically determines that the contents of the file require more than one of the specified type of PROM devices to contain the code. To copy the Human Interface, enter the following commands (shown in blue):

```
- IPPS  <CR>
iRMX I INTEL PROM PROGRAMMING SOFTWARE <version>
COPYRIGHT <years> INTEL CORP.
iRMX I/II UDI-Based IUP Interface <version>
PPS> type 27512  <CR>
PPS> init 86  <CR>
PPS> copy hi.evn(0) to p(ca08)  <CR>
----CAUTION-----PROGRAMMING THE FULL LENGTH REQUIRES MORE THAN ONE PROM.
 CHECK SUM = CB19
FIRST INSTALL THE NEW/NEXT PROM AND THEN CONTINUE.
```

When IPPS prompts you to install the next PROM, remove U2 from the PROM programmer and replace it with U3. Then enter the following:

```
CONTINUE--Y/N? y <CR>
```

When the checksum value appears on the screen, enter:

```
PPS> exit <CR>
```

Now, copy the remaining subsystems into U3 by invoking the second SUBMIT file as follows:

```
- SUBMIT copylast(evn)  <CR>
```

When the SUBMIT file ends, remove U3 from the PROM Programmer.

The two PROM devices you have removed contain the even (or low) bytes of the WORD values that compose the operating system. As you remove the PROM devices from the programmer, carefully label them; unlabeled PROM devices look very much alike. At a later time, you will place the first programmed PROM device in socket U41 and the second programmed PROM device in socket U40 of an iSBC 286/10(A) board. (Use sockets U2 and U3 of the iSBX 341 on an iSBC 286/12 board.)

Next, you need to program the U5 and U6 PROM devices to contain the odd (or high) bytes of the WORD values that compose the operating system. Insert the third PROM device (U5) into the active socket of the PROM programmer and enter the following command:

```
- SUBMIT copy1st(odd)  <CR>
```

This invokes IPPS and copies the odd bytes of the Basic I/O System into the U5 PROM device.

When the SUBMIT file ends, manually invoke IPPS to begin copying the Human Interface into U5. Again, IPPS automatically determines that the contents of the file requires more than one of the specified type of PROM devices to contain the code. To copy the Human Interface, enter the following commands (shown in blue):

```
- IPPS  <CR>
iRMX I INTEL PROM PROGRAMMING SOFTWARE <version>
COPYRIGHT <years> INTEL CORP.
iRMX I/II UDI-Based IUP Interface <version>
PPS> type 27512  <CR>
PPS> init 86  <CR>
PPS> copy hi.odd(0) to p(ca08)  <CR>
----CAUTION-----PROGRAMMING THE FULL LENGTH REQUIRES MORE THAN ONE PROM.
 CHECK SUM = CB19
FIRST INSTALL THE NEW/NEXT PROM AND THEN CONTINUE.
```

When IPPS prompts you to install the next PROM, remove U5 from the PROM programmer and replace it with U6. Then enter the following:

```
CONTINUE--Y/N? y <CR>
```

When the checksum value appears on the screen, enter:

```
PPS> exit <CR>
```

Finally, copy the remaining subsystems into U6 by invoking the second SUBMIT file as follows:

```
- SUBMIT copylast(odd)  <CR>
```

When the SUBMIT file ends, remove U6 from the PROM Programmer. The U5 and U6 PROM devices now contain the odd (or high) bytes of the WORD values that compose the system. As you remove the PROM devices from the programmer, carefully label them; unlabeled PROM devices look very much alike. At a later time, you will place the third programmed PROM device in socket U76 and the fourth programmed PROM device in socket U75 of an iSBC 286/10(A) board. (Use sockets U5 and U6 of the iSBX 341 on an iSBC 286/12 board.)

C.4.5 Starting the Operating System in ROM from the iSDM™ Monitor

The four PROM devices now contain the operating system, the iSDM monitor, and the Bootstrap Loader. Perform the following steps to start the system:

1. Place the first programmed PROM device (U2) in socket U41 of the iSBC 286/10(A) board.
2. Place the second programmed PROM device (U3) in socket U40 of the iSBC 286/10(A) board.
3. Place the third programmed PROM device (U5) in socket U76 of the iSBC 286/10(A) board.
4. Place the fourth programmed PROM device (U6) in socket U75 of the iSBC 286/10(A) board.
5. Insert the iSBC 286/10(A) board into the system chassis and apply power to the hardware.
6. Enter the following command from the iSDM monitor to activate the iRMX I system:

   ```
   .g flec:0  <CR>
   ```

   The address specified above is the root job starting address.

C.5 HARDWARE JUMPER MODIFICATIONS

To program the system into PROM devices as in the above example, the following jumpers were changed on the iSBC 286/10(A) board:

To specify 4 27512 EPROM devices, set up jumpers 62 through 91 as follows;

Default Configuration	Jumpers to Set for 27512 EPROMS
E62 - E63	E65 - E67
E70 - E72	E68 - E70
E71 - E73	E71 - E73
E75 - E76	E75 - E76
E77 - E78	E80 - E82
E85 - E87	E83 - E85
E86 - E88	E86 - E88
E90 - E91	E90 - E91

To specify a starting memory address and memory size for local memory, use primary decode option 3. The jumpers required are

Default Configuration	Jumpers for Primary Decode Option 3
E218 - E219 installed	E218 - E219 installed
E220 - E221 removed	E220 - E221 installed

To specify memory/size/justification for local memory, use secondary option 3. The jumpers required are

Default Configuration	Jumpers for Secondary Option 3
E51 - E59 removed	E51 - E59 removed
E50 - E58 removed	E50 - E58 installed
E49 - E57 installed	E49 - E57 installed

INDEX

intel®

Guide to the iRMX®
Interactive Configuration Utili
462923-0C

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative.

1. Please describe any errors you found in this publication (include page number).

____
____
____
____

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

____
____
____
____

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

____
____
____
____

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

____
____
____

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). ____________

NAME ______________________________ DATE __________
TITLE ____
COMPANY NAME/DEPARTMENT ____________________________
ADDRESS ____________________________ PHONE (   ) __________
CITY ______________ STATE ______________ ZIP CODE __________
(COUNTRY)

Please check here if you require a written reply ☐

/E'D LIKE YOUR COMMENTS . . .

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

you are in the United States, use the preprinted address provided on this form to return your ɔmments. No postage is required. If you are not in the United States, return your comments to the Intel ɿles office in your country. For your convenience, international sales office addresses are printed on e last page of this document.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 79 HILLSBORO, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
OMSO Technical Publications, MS: HF3-72
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124-9978

INTERNATIONAL SALES OFFICES

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

BELGIUM
Intel Corporation SA
Rue des Cottages 65
B-1180 Brussels

DENMARK
Intel Denmark A/S
Glentevej 61-3rd Floor
dk-2400 Copenhagen

ENGLAND
Intel Corporation (U.K.) LTD.
Piper's Way
Swindon, Wiltshire SN3 1RJ

FINLAND
Intel Finland OY
Ruosilante 2
00390 Helsinki

FRANCE
Intel Paris
1 Rue Edison-BP 303
78054 St.-Quentin-en-Yvelines Cedex

ISRAEL
Intel Semiconductors LTD.
Atidim Industrial Park
Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

ITALY
Intel Corporation S.P.A.
Milandfiori, Palazzo E/4
20090 Assago (Milano)

JAPAN
Intel Japan K.K.
Flower-Hill Shin-machi
1-23-9, Shinmachi
Setagaya-ku, Tokyo 15

NETHERLANDS
Intel Semiconductor (Netherland B.V.)
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam

NORWAY
Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten

SPAIN
Intel Iberia
Calle Zurbaran 28-IZQDA
28010 Madrid

SWEDEN
Intel Sweden A.B.
Dalvaegen 24
S-171 36 Solna

SWITZERLAND
Intel Semiconductor A.G.
Talackerstrasse 17
8125 Glattbrugg
CH-8065 Zurich

WEST GERMANY
Intel Semiconductor G.N.B.H.
Seidlestrasse 27
D-8000 Munchen

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