intel®

iRMX™ 86 RELEASE 6
DOCUMENTATION CHANGE PACKAGE:
UPDATE 3

iRMX™
86
OPERATING
SYSTEM

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

Order Number: 147540-001

iRMX™ 86 RELEASE 6
DOCUMENTATION CHANGE PACKAGE: UPDATE 3

Order Number: 147540-001

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

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

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

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

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

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

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

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

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel Corporation retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

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

BITBUS
COMMputer
CREDIT
Data Pipeline
Genius
$\overset{\Delta}{i}$
i
I^2ICE
ICE
iCS
iDBP
iDIS
iLBX
im
iMDDX
iMMX
Insite
Intel
int$_e$l
int$_e$lBOS
Intelevision
int$_e$ligent Identifier
int$_e$ligent Programming
Intellec
Intellink
iOSP
iPDS
iRMX
iSBC
iSBX
iSDM
iSXM
KEPROM
Library Manager
MCS
Megachassis
MICROMAINFRAME
MULTIBUS
MULTICHANNEL
MULTIMODULE
OpenNET
Plug-A-Bubble
PROMPT
Promware
QueX
QUEST
Ripplemode
RMX/80
RUPI
Seamless
SLD
SYSTEM 2000
UPI

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

===

iRMX™ 86 OPERATING SYSTEM RELEASE 6 CHANGE PACKAGE: UPDATE 3
147540-001

===

Purpose

The change pages in this package correct technical errors identified in the current version of the iRMX™ 86 Release 6 documentation.

Scope

The following manuals are affected by this change package:

Introduction and Operator's Reference Manual (146194-001)
Programmer's Reference Manual, Part I (146195-001)
Programmer's Reference Manual, Part II (146196-001)
iRMX™ 86 Installation and Configuration Guide (146197-001)

iRMX™ 86 Update Change Package Description

The iRMX™ 86 OPERATING SYSTEM RELEASE 6 CHANGE PACKAGE: UPDATE 3 consists of a series of corrected pages that replace the corresponding pages in your documentation. A change package for iRMX™ 86 Release 6 documentation is issued each quarter in conjunction with the iRMX™ 86 Release 6 Update Package. In addition to the change pages issued for the current update, each change package also contains an accumulation of the change pages from all previous updates.

The change pages in this package are organized into sections according to the update in which they were issued. All change pages for the current update are in a section at the front of the package. Change pages from previous updates are in succeeding sections.

Each update section begins with a blue cover and is subdivided into four segments, one for each of the iRMX™ 86 Release 6.0 documentation volumes. Each of these volume segments is identified by a yellow, pink, green, or orange cover sheet. Within each volume segment the change pages are organized in the sequence in which they occur in the volume.

The Update Revision History pages--located immediately behind the sheet you are now reading--maintains a history of all changes distributed through the iRMX™ 86 Release 6.0 Updates. This page indicates the product enhancement or the software problem report (SPR) that initiated each change. There is one Update Revision History page for each of the four iRMX™ 86 documentation volumes.

Installation Instructions

Change pages in the Update Package are accumulated from quarter to quarter. The change pages for each successive update are separated in this package by a blue cover page (similar to the sheet you are now reading). Within each update section, yellow, pink, green, and orange cover sheets segregate the change pages according to volume.

The change pages in this package are installed by removing a page from your documentation and replacing it with the corresponding page from the change package.

<u>If this is the first iRMX™ 86 Release 6.Ø Update to be installed in your documentation:</u>

1. Immediately behind the change package cover sheet (the sheet you are now reading) are four Update Revision History pages--one for each of the four volumes of iRMX 86™ Operating System documentation. Install each Update Revision History page in the front of the appropriate volume.

2. Install all of the change pages in the package. Begin with the change pages issued for Update 2. (The Update 2 change pages are located in the bottom half of the package, behind the second blue cover sheet.) After installing the Update 2 change package, install the change pages for Update 3. (The Update 3 change pages are located immediately behind the sheet you are now reading.) <u>You must install the Update 2 change package before installing the Update 3 change package</u>. If you were to install Update 2 last, you would risk replacing a current (Update 3) version of a page with an Update 2 version of the same page.

3. Fill out the Reader Comment Card--located at the bottom of the package--and mail it to Intel Corporation.

<u>If you have installed previous iRMX™ 86 Release 6.Ø Updates in your documentation</u>:

1. Immediately behind the change package cover sheet (the sheet you are reading) are four Update Revision History pages--one for each of the four volumes of iRMX 86™ Operating System documentation. In the front of each of your volumes, replace the Update History Pages from the previous update with the Update History Pages for Update 3.

2. Install only the change pages for Update 3. These change pages are in the first section at the top of the package.

3. Discard the remainder of the change pages in the change package. (These pages should already be in your documentaion if you installed the previous update.)

4. Fill out the Reader Comment Card--located at the bottom of the package--and mail it to Intel Corporation.

UPDATE REVISION HISTORY

Introduction and Operator's Reference Manual (146194-ØØ1)

Manual	Page	Initiated By	Distribution
OP	2-11/12	SPR# 1Ø2943	Update 2 (12/84)
OP	3-15/16	SPR# 1Ø3257,1Ø3133	
OP	3-83/86	SPR# 1Ø29Ø5	
OP	3-97/1ØØ	SPR# 1Ø3155	
OP	3-113/115	Addition of ZSCAN	
DV	2-7/8	SPR# 1Ø3153	
DV	A-7/8	SPR# 1Ø3151,1Ø3152, 1Ø3154	
DV	Ind-1/3	SPR# 1Ø3149,1Ø3148, 1Ø3147,1Ø315Ø	
OP	3-7/8	SPR# 1Ø3345	Update 3 (3/85)
OP	3-81/82	SPR# 1Ø3239	
OP	A-9/1Ø	SPR# 1Ø3387	
DV	iii/iv	SPR# 1Ø3353	
DV	1-1/4	SPR# 1Ø3354,1Ø337Ø	
DV	2-45/46	SPR# 1Ø3355	
DV	A-7/8	SPR# 1Ø3252	

IN=Introduction to iRMX™ 86, OP=Operator's Manual, DV=Disk Verification

UPDATE REVISION HISTORY

Programmer's Reference Manual, Part I (146195-ØØ1)

Manual	Page	Initiated By	Distribution
NU	7-13/14	SPR# 1Ø3174	Update 2, (12/84)
NU	8-3/4	SPR# 1Ø2927	
NU	12-131/132	SPR# 1Ø3173	
NU	12-149/15Ø	SPR# 1Ø3175	
NU	12-153/154	SPR# 1Ø3Ø51	
BI	8-99/1ØØ	SPR# 1Ø297Ø	
BI	8-1Ø3/1Ø6	SPR# 1Ø297Ø	
BI	F-9/1Ø	SPR# 1Ø3Ø58	
MI	Ind 15/16	Addition of 188/48 Driver	
MI	Ind 17.1	Addition of 188/48 Driver	
MI	Ind 29/3Ø	Addition of 188/48 Driver	
NU	5-3/4	SPR# 1Ø33ØØ	Update 3, (3/85)
NU	12-9/1Ø	SPR# 1Ø3323	
NU	12-21/22	SPR# 1Ø3294	
NU	12-37/4Ø	SPR# 1Ø3385	
NU	12-43/44	SPR# 1Ø3324	
NU	12-59/6Ø	SPR# 1Ø3326	
NU	12-81/82	SPR# 1Ø3328	
NU	12-95/96	SPR# 1Ø3384	
NU	12-137/138	SPR# 1Ø3299	
BI	8-11/14	SPR# 1Ø3329,1Ø3382	
BI	8-15/16	SPR# 1Ø3331	
BI	8-29/3Ø	SPR# 1Ø338Ø	
BI	8-87/88	SPR# 1Ø3383	
BI	8-97/1ØØ	SPR# 1Ø3332, 1Ø3333, 1Ø3334	
BI	8-1Ø3/1Ø6	SPR# 1Ø3335, 1Ø3336, 1Ø3337	
BI	8-1Ø9/11Ø	SPR# 1Ø3338	
BI	8-127/128	SPR# 1Ø3339	
BI	8-135/136	SPR# 1Ø334Ø,1Ø3386	
EI	7-7/8	SPR# 1Ø3398	
EI	C-1/2	SPR# 1Ø3352	
EI	Ind-3/4	SPR# 1Ø3348	

NU=Nucleus, BI=BIOS, EI=EIOS

UPDATE REVISION HISTORY

Programmer's Reference Manual, Part II (146196-ØØ1)

Manual	Page	Initiated By	Distribution
HI	8-45/48	SPR# 1Ø3171,1Ø3172	Update 2, (12/84)
UDI	2-41/42	SPR# 1Ø3Ø59	
UDI	2-53/54	SPR# 1Ø3Ø6Ø	
DD	6-3/6	SPR# 1Ø3Ø54	
PT	3-7/8	SPR# 1Ø3121	
TH	3-1/2	SPR# 1Ø3121	
BL	2-9/1Ø	SPR# 1Ø3Ø81	
AL	2-11/14	SPR# 1Ø3342,1Ø3351	Update 3, (3/85)
AL	2-21/22	SPR# 1Ø3343	
AL	2-29/3Ø	SPR# 1Ø3344	
HI	5-3/4	SPR# 1Ø3212	
HI	B-9/1Ø	SPR# 1Ø33Ø4	
UD	2-5/8	SPR# 1Ø332Ø	
UD	2-53/54	Fix Update 2 Error	
DD	Ind-3/4	SPR# 1Ø3349, 1Ø335Ø	
PT	6-1	SPR# 1Ø3176	
TH	2-1/2	SPR# 1Ø3368	
TH	3-1/2	SPR# 1Ø3369	

AL=Application Loader, HI=Human Interface, UD=UDI, DD=Device Drivers
PT=Programming Techniques, TH=Terminal Handler, DB=Debugger,
CA=Crash Analyzer, SD=System Debugger, BT=Bootstrap Loader

UPDATE REVISION HISTORY

iRMX™ 86 Installation and Configuration Guide (146197-ØØ1)

Manual	Page	Initiated By	Distribution
IG	6-9/1Ø	SPR# 1Ø2944	Update 2, (12/84)
IG	6-15/16	SPR# 1Ø3Ø5Ø	
IG	8-1/2	Addition of 188/48 Driver	
IG	1Ø-5/12	SPR# 1Ø3Ø7Ø,1Ø3211,1Ø3255 1Ø3Ø85,1Ø3169	
IG	D-3/6	SPR# 1Ø2959,1Ø2958	
IG	E-1	SPR# 1Ø2945	
IG	F-1/2	Addition of 188/48 Driver	
IG	Ind-1/2	Addition of 188/48 Driver	
CG	2-15/16	Addition of 188/48 Driver	
CG	4-1/2	Addition of 188/48 Driver	
CG	1Ø-1/2	Addition of 188/48 Driver, SPR# 1Ø296Ø	
CG	1Ø-152.1/ 152.12	Addition of 188/48 Driver	
CG	15-7/1Ø	SPR# 1Ø3Ø74	
CG	18-3/4	Addition of 188/48 Driver	
CG	B-3/4	Addition of 188/48 Driver	
CG	B-11/14	Addition of 188/48 Driver	
IG	6-15/18	SPR# 1Ø3272,1Ø3372	Update 3, (3/85)
CG	1Ø-1/2	Fix Update 2 Error	
CG	1Ø-33/34	SPR# 1Ø3363	
CG	xxv/xxviii	Addition of 217 Driver, Addition of 226 Driver	
CG	E-1/2Ø	Addition of 217 Driver	
CG	F-1/1Ø	Addition of 226 Driver	

IG=Installation, CG=Configuration, MI=Master Index

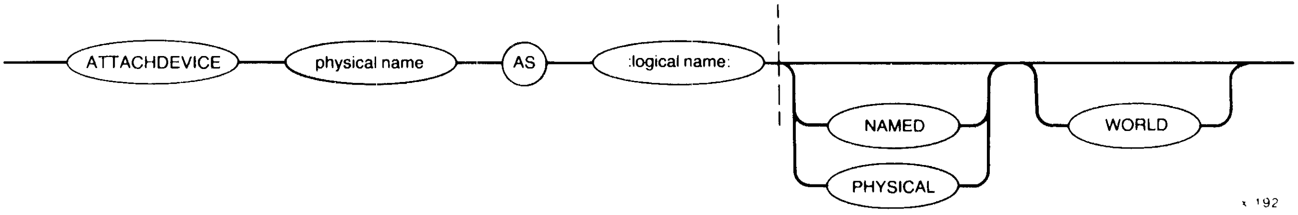
iRMX™ 86 Release 6.Ø Change Package: Update 3

Change Pages for:

iRMX™ 86 Introduction and Operator's Reference Manual (146194-ØØ1)

ATTACHDEVICE

This command attaches a physical device to the Operating System and associates a logical name with the device. The command catalogs the logical name in the root object directory, making the logical name accessible to all users. The format of the command is as follows:



INPUT PARAMETERS

physical name

Physical device name of the device to be attached to the system. This name must be the name used in one of the Basic I/O System's Device Unit Information Blocks (DUIB), as defined at system configuration time (see Table 3-2).

AS

Preposition; required for the command.

:logical name:

A 1-1Ø character string that represents the logical name to be associated with the device. Colons surrounding the logical name are optional; however, if you use colons, you must use matching colons.

NAMED

Specifies that the volume mounted on the device is already formatted for NAMED files. Examples of volumes that can contain named files are diskettes or hard disk platters. If neither NAMED nor PHYSICAL are specified, NAMED is the default. See the FORMAT command in this chapter for a further description of NAMED files.

PHYSICAL

Specifies that the volume mounted on the logical device is considered to be a single, large file. Examples include line printers and terminals. See the FORMAT command in this chapter for a further description of PHYSICAL volumes.

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

DESCRIPTION

ATTACHDEVICE attaches a device to the system and catalogs a logical name for it in the root job's object directory. The logical name is the means by which all users can access the device. Devices must have their characteristics listed in the Basic I/O System's Device Unit Information Block (DUIB) at configuration time before they can be attached with the ATTACHDEVICE command.

Table 3-2 and Table 3-3 list the physical device names normally used with the Basic I/O System. Your system might support a subset of these devices or it might support devices not listed. If it supports the devices listed, it might support them under different names. Therefore, consult the person who configured your system to determine the correct device names for your system.

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

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

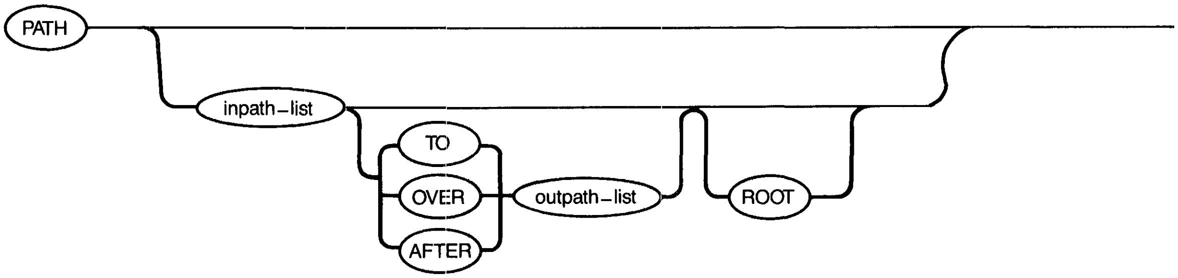
When the device attachment is completed, the ATTACHDEVICE command displays the following message:

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

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

PATH

This command lists the pathname of a data file or directory.



X-941

INPUT PARAMETERS

inpath-list — The list of files whose pathnames you want to know. The default inpath-list file directory is the current working directory (:$:).

ROOT — Specifies that the pathname should start from the root directory of whatever device holds the file or directory.

OUTPUT PARAMETERS

TO — Writes the pathnames of the input files to the specified output files. The specified output file or files should not already exist. If they do, PATH displays the following message:

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

Enter Y, y, R, or r if you wish to write over the existing file. Enter an N (upper or lower case) or a carriage return alone if you do not wish to overwrite the existing file. In the latter case, the PATH command will pass over the corresponding input file, and will attempt to write the pathname of the next input file to the corresponding output file.

If you specify multiple input files and a single output file, PATH appends the remaining input file pathnames to the end of the output file.

OVER — Writes the input file pathname over (replaces) the existing output files on a one-for-one basis, regardless of file size. If an output file does not already exist, the corresponding input file pathname is written to a new file with the corresponding output file name. If you specify multiple input files and a single output file, PATH appends the remaining input file pathnames to the end of the output file.

AFTER — Appends the input file pathname(s) to the current data in the existing output file or files. If the output file does not already exist, all listed input file pathnames will be concatenated into a new file with the listed output file name.

outpath-list — One or more pathnames for the output files.

DESCRIPTION

This command is useful for finding where you may be located within the file structure. The command gives the following listing when it is invoked with no input file listing:

```
--PATH
:sd:user/world
```

Table A-1. iRMX↓ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N B E L H	Meaning
Programmer Errors (continued)			
8ØØ4H	E$PARAM	* * * * *	A parameter which is neither a token nor an offset has an invalid value.
8ØØ5H	E$BAD$CALL	* *	The I/O System code has been damaged, probably due to a bug in an application task. Recovery is not possible.
8ØØ6H	E$ARRAY$-BOUNDS	*	Hardware or software has detected an array overflow.
8ØØ7H	E$NDP$-STATUS	*	An 8Ø87 Numeric Processor Extension error has been detected; Operating System extensions can return the status of the 8Ø87 to the exception handler.
8ØØ8H	E$ILLEGAL$-OPCODE	*	The iAPX 186 or 286 processor tried to execute an invalid instruction. (Software interrupt 6)
8ØØ9H	E$EMULATOR$-TRAP	*	The iAPX 186 or 286 processor tried to execute an ESC instruction with the "emulator" bit set in the relocation register (iAPX 186) or the machine status word (iAPX 286).
8ØØAH	E$INTERRUPT$-TABLE$LIMIT	*	An iAPX 286 LIDT instruction changed the interrupt table limit to a value between 2ØH and 42H.
8ØØBH	E$CPUXFER$-DATA$LIMIT	*	For an iAPX 286 processor, the processor extension data transfer exceeded the offset of ØFFFFH in a segment.
8ØØCH	E$SEG$WRAP$-AROUND	*	For an iAPX 286 processor, either a word operation attempted a segment wraparound at offset ØFFFFH; or a PUSH, CALL, or INT instruction attempted to execute while SP=1.

N	Nucleus Reference Manual	L	Loader Reference Manual
B	Basic I/O System Ref Manual	H	Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		

Table A-1. iRMX↓ 86 Condition Codes (continued)

Hex. Value	Mnemonic	Manuals N	B	E	L	H	Meaning
							Programmer Errors (continued)
8017H	E$CHECK$EXCEPTION	*					A Pascal task has exceeded the bounds of a CASE statement.
8021H	E$NOUSER		*	*		*	No default user.
8022H	E$NOPREFIX		*	*		*	No default prefix.
8040H	E$NOT$LOG$NAME			*		*	Specified object is not a device connection or file connection.
8041H	E$NOT$DEVICE			*			A token parameter referred to an existing object that is not, but should be, a device connection.
8042H	E$NOT$CONNECTION			*			A token parameter referred to an existing object that is not, but should be, a file connection.
8060H	E$JOB$PARAM				*	*	The maximum job-size specified is less than the minimum job-size.
8080H	E$PARSE$TABLES					*	There is an error in the internal parse tables.
8081H	E$JOB$TABLES					*	An internal Human Interface table was overwritten, causing it to contain an invalid value.
8085H	E$ERROR$OUTPUT					*	The command invoked by C$SEND$COMMAND includes a call to C$SEND$EO$RESPONSE, but the command connection does not permit C$SEND$EO$RESPONSE calls.

N	Nucleus Reference Manual	L	Loader Reference Manual
B	Basic I/O System Ref Manual	H	Human Interface Reference Manual
E	Extended I/O Sys Ref Manual		

PREFACE

This manual documents the Disk Verification Utility, a software tool that runs as a Human Interface command, verifying and modifying the data structures of iRMX 86 named and physical volumes. The manual describes the utility invocation and contains detailed descriptions of all utility commands. Also, because users must be familiar with the structure of iRMX 86 volumes to use the Disk Verification Utility features intelligently, the manual contains an appendix that describes the structure of iRMX 86 named volumes.

READER LEVEL

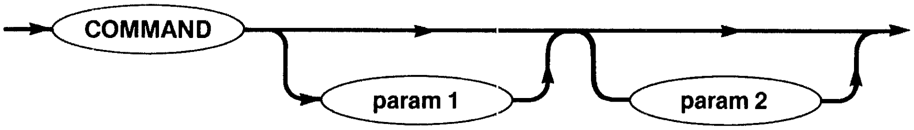
This manual is intended for system programmers who have had experience in examining actual volume information. It does not attempt to teach the user the proper procedures for examining and editing volume information.

NOTATIONAL CONVENTIONS

This manual uses the following conventions to illustrate syntax.

UPPERCASE	Uppercase information must be entered exactly as shown. You can, however, enter this information in uppercase or lowercase.
lowercase	Lowercase fields contain variable information. You must enter the appropriate value or symbol for variable fields.
underscore	In examples of dialog at the terminal, user input is underscored to distinguish it from system output.
<variable>	Whenever an error message or the output resulting from a DISKVERIFY command contains a variable part, that variable part is enclosed in angle brackets < >.

Also, this manual uses the "railroad track" schematic to illustrate the syntax of the disk verification commands. This syntax consists of what looks like an aerial view of a model railroad setup, with syntactic elements scattered along the track. To interpret the command syntax, you start at the left side of the schematic, follow the track through all the syntactic elements you desire (sharp turns and backing up are not allowed), and exit at the right side of the schematic. The syntactic elements that you encounter, separated by spaces, comprise a valid command. For example, a command that consists of a command name and two optional parameters would have the following schematic representation:



x-285

You could enter this command in any of the following forms:

```
    COMMAND
    COMMAND param1
    COMMAND param2
    COMMAND param1 param2
```

The arrows indicate the possible flow through the tracks; they are omitted in the remainder of the manual.

CHAPTER 1
INVOKING THE DISK VERIFICATION UTILITY

In the process of using an iRMX 86 application system, you may have occasion to store data on secondary storage devices, sometimes large amounts of data. Due to the nature of secondary storage devices, unforseen circumstances such as power irregularities or accidental reset may destroy information on these devices, causing them to be inaccessible to your iRMX 86 system. In some cases, the loss of only a small amount of data can render an entire volume, such as a disk, useless.

In such cases, it is desirable to have a mechanism to examine and modify the damaged volume. This mechanism would allow you to determine how much of the information on the volume was damaged. It would also allow you to recreate file structures on the damaged volume so that you could salvage some of the valid data. The iRMX 86 disk verification utility is a tool that allows you to perform these functions.

The disk verification utility verifies the data structures of iRMX 86 physical and named volumes. It can also be used to reconstruct the free fnodes map, the volume free space map, and the bad blocks map of the volume and perform absolute editing.

You can use the disk verification utility in one of two ways:

- o As a single command which verifies the structures of a volume and returns control to the Human Interface.

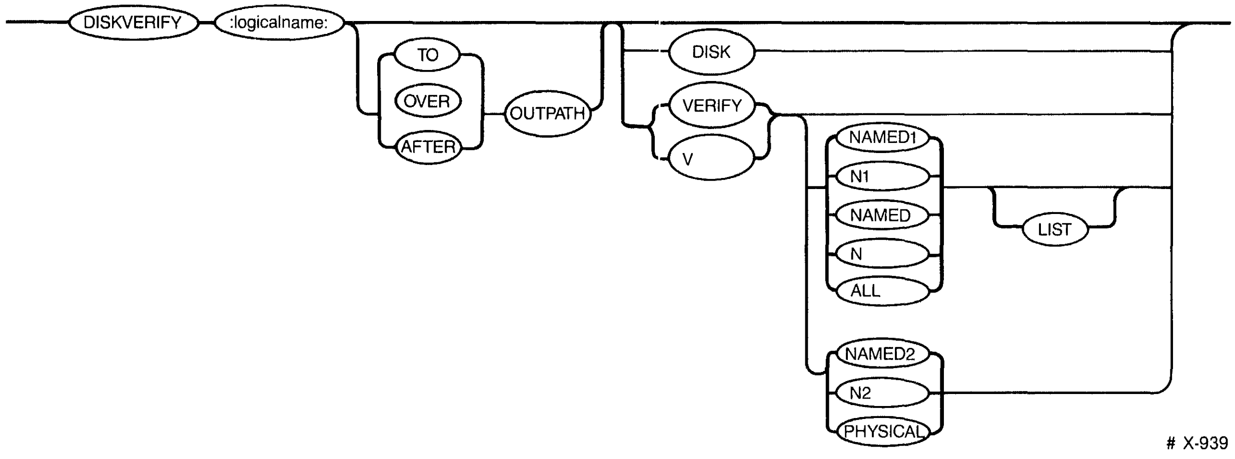
- o As an interactive program which allows you to check and modify information on the volume by entering individual disk verification commands.

To take full advantage of the capabilities of the disk verification utility, you must be familiar with the structure of iRMX 86 named volumes. Appendix A contains detailed information about the volume structure. If you are unfamiliar with the iRMX 86 volume structure, you should avoid using the individual disk verification commands. When used carelessly, these commands can make your volumes unusable.

However, even if you know nothing about iRMX 86 volume structures, you can still use the utility as a single command to verify that the data structures on an iRMX 86 volume are valid.

INVOCATION

The format of the Human Interface command used to invoke the disk verification utility is as follows:



where:

:logical-name: Logical name of the secondary storage device containing the volume.

TO Copies the output from the disk verification utility to the specified file. If no preposition is specified, TO :CO: is the default.

OVER Copies the output from the disk verification utility over the specified file.

AFTER Appends the output from the disk verification utility to the end of the specified file.

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

DISK

Displays the attributes of the volume being verified.

If you specify this parameter, the utility performs the disk function and returns control to you at the Human Interface level. You can then enter any Human Interface command provided that the device verified is not the system device. Refer to the description of the DISK command in Chapter 2 for more information. Any parameter after this one is ignored.

VERIFY or V

Performs a verification of the volume. This verification function and the associated options are described in detail in the "VERIFY Command" section of Chapter 2. If you specify this parameter and omit the options, the utility performs the NAMED verification.

If you specify this parameter, the utility performs the verification function and returns control to you at the Human Interface level. You can then enter any Human Interface command if the device is not the system device (:sd:).

If you omit this parameter and the DISK parameter, the utility displays a header message and the utility prompt (*). You can then enter any of the disk verification commands listed in Chapter 2.

NAMED1 or N1

VERIFY option that applies to named volumes only. This option checks the fnodes of the volume to ensure that they match the directories in terms of file type and file heirarchy. This option also checks the information in each fnode to ensure that it is consistent. Refer to the description of the VERIFY command in Chapter 2 for more information.

NAMED or N

VERIFY option that performs both the NAMED1 and NAMED2 verification functions on a named volume. If you omit the VERIFY option, NAMED is the default option.

ALL

VERIFY option that applies to both named and physical volumes. For named volumes, this option performs both the NAMED and PHYSICAL verification functions. For physical volumes, this option performs the PHYSICAL verification function.

NAMED2 or N2

VERIFY option that applies to named volumes only. This option checks the allocation of fnodes on the volume, checks the allocation of space on the volume, and verifies that the fnodes point to the correct locations on the volume. Refer to the description of the VERIFY command in Chapter 2 for more information.

PHYSICAL — VERIFY option that applies to both named and physical volumes. This option reads all blocks on the volume and checks for I/O errors.

LIST — VERIFY option that you can use with those VERIFY parameters that, either explicitly or implicitly, specify the NAMED1 parameter. When you use this option, the file information generated by VERIFY is displayed for every file on the volume, even if the file contains no errors. Refer to the description of the VERIFY command in Chapter 2 for more information.

OUTPUT

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

```
iRMX 86 DISK VERIFY UTILITY, Vx.x
Copyright <year> Intel Corporation
```

where Vx.x is the version number of the utility. If you specify the VERIFY or V parameter in the DISKVERIFY command, the utility performs a verification of the volume and copies the verification information to the console (or to the file specified by the outpath parameter). The verification information is the same as that produced by the VERIFY utility command. Refer to the description of the VERIFY command in Chapter 2 for a description of the verification output. After generating the verification output, the utility returns control to the Human Interface, which prompts you for more Human Interface commands. The following is an example of such a DISKVERIFY command:

```
-DISKVERIFY :F1: VERIFY NAMED2
iRMX 86 DISK VERIFY UTILITY ,  Vx.x
Copyright <year> Intel Corporation

DEVICE NAME = wfdØ       : DEVICE SIZE = ØØØ3E9ØØ : BLOCK SIZE = ØØ8Ø

'NAMED2'  VERIFICATION
    BIT MAPS O.K.

-
```

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

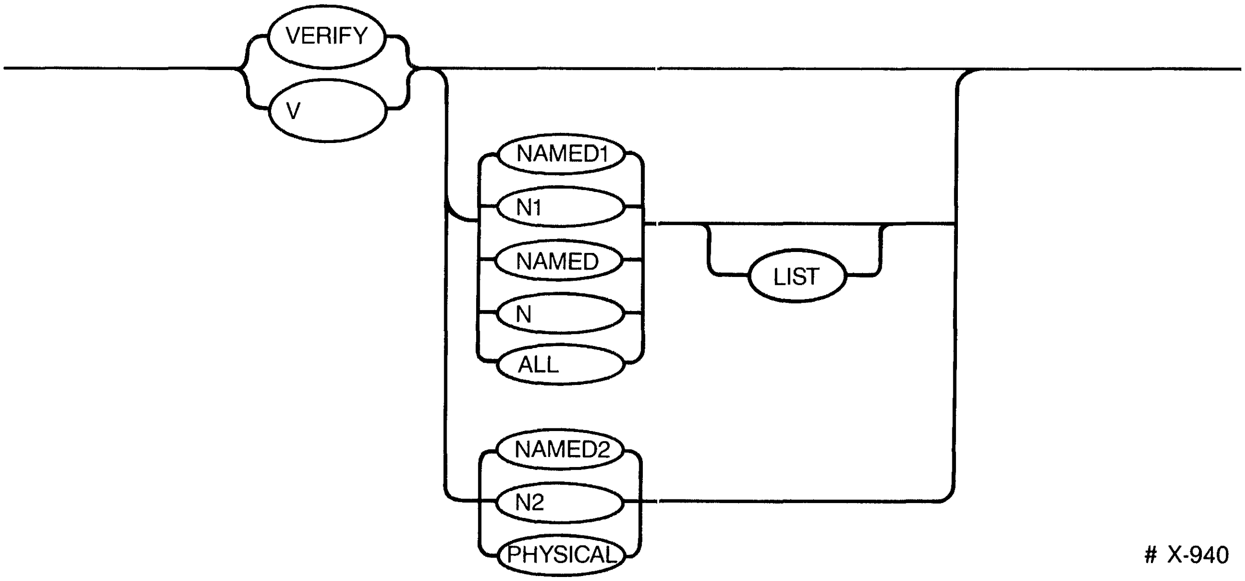
```
-DISKVERIFY :F1:
iRMX 86 DISK VERIFY UTILITY ,  Vx.x
Copyright <year> Intel Corporation
*
```

EXAMPLE (continued)

```
    *SUBSTITUTEWORD<cr>

    ØØØØ: AØBØ - ØØØØ<cr>
    ØØØ2: 8Ø7Ø - <cr>
    ØØØ4: E511 - <cr>
    ØØØ6: FFFF - 3111<cr>
    ØØØ8: FFFF - .<cr>

    *SUBSTITUTEWORD 35<cr>

    ØØ35: ØØØØ - E6FF<cr>
    ØØ37: ØØØØ - E6AB<cr>
    ØØ39: ØØØØ - .<cr>

    *
```

VERIFY COMMAND

This command checks the structures on the volume to determine whether the volume is properly formatted. You can abort this command by typing a CONTROL-C (press the CONTROL key, and while holding it down, press the C key). The format of the VERIFY command is:



INPUT PARAMETERS

NAMED1 or N1

Checks named volumes to ensure that the information recorded in the fnodes is consistent and matches the information obtained from the directories themselves. VERIFY performs the following operations during a NAMED1 verification:

- Checks fnode numbers in the directories to see if they correspond to allocated fnodes.
- Checks the parent fnode numbers recorded in the fnodes to see if they match with the information recorded in the directories.
- Checks the fnodes against the files to determine if the fnodes specify the proper file type.
- Checks the POINTER(n) structures of long files to see if the indirect blocks accurately reflect the number of blocks used by the file.
- Checks each fnode to see if the TOTAL SIZE, TOTAL BLKS, and THIS SIZE fields are consistent.
- Checks the bad blocks file to see if the blocks in the file correspond to the blocks marked as "bad" on the volume.

If the formatting program is unable to provide this information, it places an ASCII space in this field.

- The next two bytes contain a two-digit ASCII sequence number which is incremented by the formatting program each time the formatting program changes in a way that affects the volume format. The Release 4 FORMAT Human Interface command places the characters "ØØ" in this field.

- The right-most three bytes of the field contain a three-digit ASCII number specifying the version of the Basic I/O System that was used in formatting the volume (for example, the characters "Ø3Ø" would indicate version 3.Ø). If the formatting program is unable to obtain this information, it places ASCII spaces in this field.

DEVICE$SPECIAL(8) Reserved for special device-specific information. When no device-specific information exists, this field must contain zeros. If the device is a Winchester disk with an iSBC 215 controller or if the device is a disk with an iSBC 22Ø controller, the iRMX 86 Operating System imposes a structure on this field and supplies the following information:

```
SPECIAL                STRUCTURE(
CYLINDERS              WORD,
FIXED                  BYTE,
REMOVABLE              BYTE,
SECTORS                BYTE,
SECTOR_SIZE            WORD,
ALTERNATES             BYTE);
```

where:

CYLINDERS	Total number of cylinders on the drive.
FIXED	Number of heads on the fixed disk or Winchester disk.
REMOVABLE	Number of heads on the removable disk cartridge.
SECTORS	Number of sectors in a track.
SECTOR_SIZE	Sector size, in bytes.
ALTERNATES	Number of alternate cylinders.

The remainder of the Volume Label (bytes 44Ø through 511) is reserved and must be set to zero.

INITIAL FILES

Any mechanism that formats iRMX 86 named volumes must place seven files on the volume during the format process. These seven files are the fnode file, the volume label file, the volume free space map file, the free fnodes map file, the bad blocks file, the root directory, and the space accounting file. The first of these files, the fnode file, contains information about all of the files on the volume. The general structure of the fnode file is discussed first. Then all of the files are discussed in terms of their fnode entries and their functions.

FNODE FILE

A data structure called a file descriptor node (or fnode) describes each file in a named file volume. All the fnodes for the entire volume are grouped together in a file called the fnode file. When the I/O System accesses a file on a named volume, it examines the iRMX 86 Volume Label (described in the previous section) to determine the location of the fnode file, and then examines the appropriate fnode to determine the actual location of the file.

When a volume is formatted, the fnode file contains seven allocated fnodes and any number of un-allocated fnodes. The original number of un-allocated fnodes depends on the FILES parameter of the FORMAT command. These allocated fnodes represent the fnode file, the volume label file, the volume free space map file, the free fnodes map file, the bad blocks file, the root directory, and the space accounting file. Later sections of this chapter describe these files. The size of the fnode file is determined by the number of fnodes that it contains. The number of fnodes in the fnode file also determines the number of files that can be created on the volume. The number of files is set when you format the storage medium.

The structure of an individual fnode in a named file volume is as follows:

```
       DECLARE
                FNODE          STRUCTURE(
                       FLAGS                WORD,
                       TYPE                 BYTE,
                       GRAN                 BYTE,
                       OWNER                WORD,
                       CR$TIME              DWORD,
                       ACCESS$TIME          DWORD,
                       MOD$TIME             DWORD,
                       TOTAL$SIZE           DWORD,
                       TOTAL$BLKS           DWORD,

                       POINTR(4Ø)           BYTE,

                       THIS$SIZE            DWORD,
                       RESERVED$A           WORD,
                       RESERVED$B           WORD,
                       ID$COUNT             WORD,

                       ACC(9)               BYTE,
                       PARENT               WORD,
                       AUX(*)               BYTE);
```

iRMX™ 86 Release 6.Ø Change Package: Update 3

Change Pages for:

iRMX™ 86 Programmer's Reference Manual, Part I (146195-ØØ1)

MOVEMENT OF MEMORY BETWEEN JOBS

When a task tries to create a segment (or an object of any other type), and the unallocated part of its job's pool is not sufficient to satisfy the request, the Nucleus tries to borrow more memory from the job's parent (and then, if necessary, from its parent's parent, and so on). Such borrowing increases the pool size of the borrowing job and is thus restricted by the pool maximum attribute of the borrowing job.

When a job is deleted, the memory in its pool becomes unallocated, and access to it is given back to the parent job. The smallest contiguous piece of memory that a job may borrow from its parent is a configuration parameter. The subject of configuration is covered in the iRMX 86 CONFIGURATION GUIDE.

Observe that, if a job has equal pool minimum and pool maximum attributes, then its pool is fixed at that common value. This means that, once it has this amount, the job may not borrow memory from its parent.

MEMORY ALLOCATION

The memory pool of a job consists of two classes of memory: allocated and unallocated. Memory in a job is unallocated unless it has been requested, either explicitly or implicitly, by tasks in the job or unless it is on loan to a child job. A task's request for memory is explicit when it calls the CREATE$SEGMENT system call. A request is implicit when the task attempts to create any type of object other than a segment.

The Nucleus borrows small amounts of memory from a job's pool each time a task in that job creates an object. This memory is needed for bookkeeping purposes. When the object is deleted, the borrowed memory is returned to the pool. Appendix B lists these memory requirements.

When a task no longer needs a segment, it can return the segment to the unallocated part of the job's pool by using the DELETE$SEGMENT system call. Figure 5-2 shows how memory "moves".

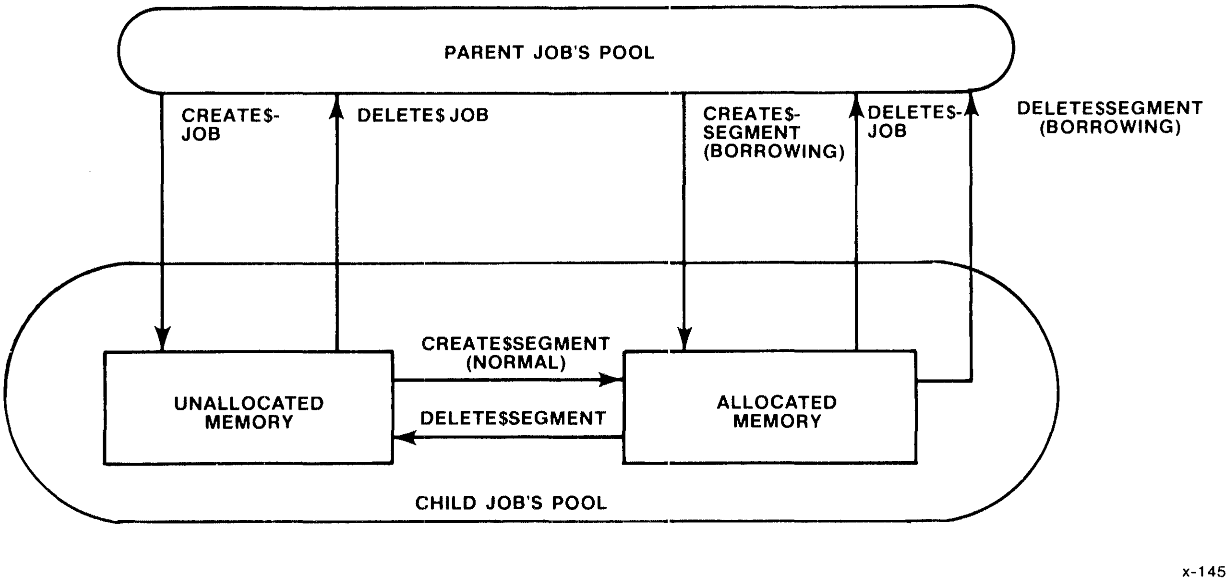


Figure 5-2. Memory Movement Diagram

SYSTEM CALLS FOR SEGMENTS

The following system calls manipulate segments:

- CREATE$SEGMENT --- creates a segment and returns a token for it.
- DELETE$SEGMENT --- returns a segment to the pool from which it was allocated.
- GET$SIZE --- returns the size, in bytes, of a segment.
- SET$POOL$MIN --- enables a task to change the pool minimum attribute of its job's pool.
- GET$POOL$ATTRIB --- returns the following memory pool attributes of the calling task's job: pool minimum, pool maximum, initial size, number of allocated paragraphs, and number of available paragraphs.

CONDITION CODES

E$OK — No exceptional conditions.

E$BUSY — Another task currently has access to the protected data.

E$CONTEXT — The calling task currently has access to the region in question.

E$EXIST — The region parameter is not a token for an existing object.

E$NOT$CONFIGURED — This system call is not part of the present configuration.

E$TYPE — The region parameter is a token for an object that is not a region.

ALTER$COMPOSITE

The ALTER$COMPOSITE system call replaces components of composite objects.

CAUTION

> Composite objects require the creation of extension objects. Jobs that create extension objects cannot be deleted until all the extension objects are deleted. Therefore you should avoid creating composite objects in Human Interface applications. If a Human Interface application creates extension objects, the application cannot be deleted asynchronously (via a CTRL/c entered at a terminal).

```
    CALL RQ$ALTER$COMPOSITE(extension, composite, component$index,
                            replacing$obj, except$ptr);
```

INPUT PARAMETERS

extension — A TOKEN for the extension type object corresponding to the composite object being altered.

composite — A TOKEN for the composite object being altered.

component$index — A WORD whose value specifies the location (starting at 1) in the component list of the component to be replaced.

replacing$obj — A TOKEN for the replacement component object or zero, which represents no object.

OUTPUT PARAMETER

except$ptr — A POINTER to a WORD to which the iRMX 86 Operating System will return the condition code generated by this system call.

max$tasks

A WORD that specifies the maximum number of tasks that can exist simultaneously in the new job.

- If not ØFFFFH, it contains the maximum number of tasks that can exist simultaneously in the new job.
- If ØFFFFH, it indicates that there is no limit to the number of tasks that tasks in the new job can create.
- It cannot be zero. A value of ØH will produce the E$LIMIT exception.

max$priority

A BYTE that sets an upper limit on the priority of the tasks created in the new job.

- If not zero, it contains the maximum allowable priority of tasks in the new job. If max$priority exceeds the maximum priority of the parent job, an E$LIMIT error occurs.
- If zero, it indicates that the new job is to inherit the maximum priority attribute of its parent job.

except$handler

A POINTER to a structure of the following form:

```
STRUCTURE(
    EXCEPTION$HANDLER$PTR      POINTER,
    EXCEPTION$MODE             BYTE);
```

If exception$handler$ptr is not zero, then it is a POINTER to the first instruction of the new job's own exception handler. If exception$handler$ptr is zero, the new job's exception handler is the system default exception handler. In both cases, the exception handler for the new task becomes the default exception handler for the job. The exception$mode indicates when control is to be passed to the exception handler. It is encoded as follows:

Value	When Control Passes To Exception Handler
Ø	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

job$flags

A WORD containing information that the Nucleus needs to create and maintain the job. The bits (where bit 15 is the high-order bit) have the following meanings:

<u>bit</u> <u>meaning</u>

15-2 reserved.

1 If Ø, then whenever a task in the new job or any of its descendent jobs makes a Nucleus system call, the Nucleus will check the parameters for validity.

If 1, the Nucleus will not check the parameters of Nucleus system calls made by tasks in the new job. However, if any ancestor of the new job has been created with this bit set to Ø, there will be parameter checking for the new job.

Ø reserved.

task$priority

A BYTE that controls the priority of the new job's initial task.

- If not zero, it contains the priority of the new job's initial task. If the task$priority parameter is greater (numerically smaller) than the new job's maximum priority attribute, an E$PARAM error occurs.

- If zero, it indicates that the new job's initial task is to have a priority equal to the new job's maximum priority attribute.

start$address

A POINTER to the first instruction of the new job's initial task (the task created with the job).

data$seg

A WORD or SELECTOR that specifies which data segment the new job's initial task is to use.

- If not zero, it contains the base address of the data segment of the new job's initial task.

- If zero, it indicates that the new job's initial task assigns its own data segment. Refer to the iRMX 86 CONFIGURATION GUIDE for more information about data segment allocation.

CONDITION CODES

E$OK No exceptional conditions.

E$LIMIT The calling task's job has already reached its object limit.

E$MEM The memory available to the calling task's job is not sufficient to create a semaphore.

E$NOT$CON-FIGURED This system call is not part of the present configuration.

E$PARAM At least one of the following is true:

- The initial$value parameter is larger than the maximum$value parameter.
- The maximum$value parameter is Ø.

CREATE$TASK

CREATE$TASK creates a task.

```
task = RQ$CREATE$TASK (priority, start$address, data$seg, stack$ptr,
                       stack$size, task$flags, except$ptr);
```

INPUT PARAMETERS

priority — A BYTE that specifies the priority of the new task.

- If not zero, it contains the priority of the new task. The priority parameter must not exceed the maximum allowable priority of the calling task's job. If it does, an E$PARAM error occurs.
- If zero, it indicates that the new task's priority is to equal the maximum allowable priority of the calling task's job.

start$address — A POINTER to the first instruction of the new task.

data$seg — A WORD or SELECTOR that specifies the new task's data segment.

- If not zero, the WORD contains the base address of the new task's data segment.
- If zero, the WORD indicates that the new task assigns its own data segment. Refer to the iRMX 86 CONFIGURATION GUIDE for further information on data segment allocation.

stack$ptr — A POINTER that specifies the location of the stack for the new task.

- If the base portion is not zero, the Nucleus uses the sum of the offset portion and the stacksize parameter (declared during the call to CREATE$TASK) as the value of the SP register (the stackpointer).
- If the base portion is zero, the Nucleus allocates a stack to the new task. The length of the stack is equal to the value of the stack$size parameter.

stack$size

A WORD containing the size, in bytes, of the new task's stack segment. The stack size must be at least 16 bytes. The Nucleus increases specified values that are not multiples of 16 up to the next higher multiple of 16.

The stack size should be at least 3ØØ bytes if the new task is going to make Nucleus system calls. Refer to the iRMX 86 PROGRAMMING TECHNIQUES manual for further information on assigning stack sizes.

If you set the stack$ptr parameter to indicate a user-provided stack, setting the stack$size parameter causes the Nucleus to fill the user-provided stack with special characters which the iRMX 86 Debugger uses to detect stack overflow. Because of this situation, never specify a stack$size value that is larger than size of the user-provided stack.

task$flags

A WORD containing information that the Nucleus needs to create and maintain the task. The bits (where bit 15 is the high-order bit) have the following meanings:

Bits	Meaning
15-1	Reserved bits which should be set to zero
Ø	If one, the task contains floating-point instructions. These instructions require the NPX component for execution If zero, the task does not contain floating-point instructions

OUTPUT PARAMETERS

task

A TOKEN to which the Operating System will return a token for the new task.

except$ptr

A POINTER to a WORD to which the iRMX 86 Operating System will return the condition code generated by this system call.

DESCRIPTION

The CREATE$TASK system call creates a task and returns a token for it. The new task counts as one against the object and task limits of the calling task's job. Attributes of the new task are initialized upon creation as follows:

- priority: as specified in the call.
- execution state: ready.
- suspension depth: Ø.
- containing job: the job which contains the calling task.
- exception handler: the exception handler of the containing job.
- exception mode: the exception mode of the containing job.

EXAMPLE

```
/*************************************************************************
 *  This example illustrates how the CREATE$TASK system call can be     *
 *  used.                                                                *
 *************************************************************************/

    $INCLUDE(:F1:SAMPLE.EXT);     /* Declares all system calls */

    TASK_CODE: PROCEDURE EXTERNAL;
    END TASK_CODE;

    DECLARE TOKEN                 LITERALLY 'SELECTOR';
                                  /* if your PL/M compiler does not
                                     support this variable type,
                                     declare TOKEN a WORD */
    DECLARE task$token            TOKEN;
    DECLARE priority$level$66     LITERALLY '66';
    DECLARE start$address         POINTER;
    DECLARE data$seg              WORD;
    DECLARE stack$pointer         POINTER;
    DECLARE stack$size$512        LITERALLY '512'; /* new task's stack
                                                      size is 512 bytes */
    DECLARE task$flags            WORD;
    DECLARE status                WORD;

SAMPLE_PROCEDURE:
    PROCEDURE;
    start$address = @TASK_CODE;   /* first instruction of the new task */
    data$seg = Ø;                 /* task sets up own data segment */
    stack$pointer = Ø;            /* automatic stack allocation  */
    task$flags = Ø;               /* designates no floating-point
                                     instructions */
         o
         o     Typical PL/M-86 Statements
         o
```

EXAMPLE

See the example in section "The Initialization Part" of Chapter 11.

CONDITION CODES

E$OK — No exceptional conditions.

E$CONTEXT — At least one of the following is true:

- The extension type does not match the composite parameter.
- One or both of the extension or composite parameters is not a token for an existing object.
- One or both of the extension or composite parameters is a token for an object that is not of the correct type.

E$MEM — The memory available to the calling task's job is not sufficient to complete this operation.

E$NOT$CONFIGURED — This system call is not part of the present configuration.

DELETE$EXTENSION

The DELETE$EXTENSION system call deletes an extension object and all composites of that type.

CAUTION

Jobs that create extension objects cannot be deleted until the extension object is deleted. Therefore, you should avoid creating extension objects in Human Interface applications. If a Human Interface application creates extension objects, the application cannot be deleted asynchronously (via a CTRL/c entered at a terminal).

```
    CALL RQ$DELETE$EXTENSION(extension, except$ptr);
```

INPUT PARAMETER

extension	A TOKEN for the extension object to be deleted.

OUTPUT PARAMETER

except$ptr	A POINTER to a WORD to which the iRMX 86 Operating System will return the condition code generated by this system call.

DESCRIPTION

The DELETE$EXTENSION system call deletes the specified extension object type and all composite objects of that type. This makes the corresponding type code available for reuse.

If a deletion mailbox was specified when the extension type was created, then all of the composite objects created by the extension type to be deleted are sent to that deletion mailbox. In this case, this call will not be completed until all of the composite objects have been deleted.

If the extension type has no deletion mailbox, the composite objects created by the extension type to be deleted are deleted without informing the type manager.

```
        o
        o     Typical PL/M-86 Statements
        o

/************************************************************************
 *  The calling task has created a task (whose code is labeled          *
 *  TASK_CODE) which is not an interrupt task.  When this task is no    *
 *  longer needed, it may be deleted by any task that knows its token.  *
 ************************************************************************/

    CALL RQ$DELETE$TASK          (task$token,
                                 @status);
        o
        o     Typical PL/M-86 Statements
        o

END SAMPLE_PROCEDURE;
```

CONDITION CODES

E$OK — No exceptional conditions.

E$CONTEXT — The task parameter is a token for an interrupt task.

E$EXIST — One of the following conditions has occurred:

- The task parameter is not a token for an existing object.
- The task parameter represents a task whose job is being deleted.
- More than one task is trying to delete a task which is in a region.

E$NOT$CON-FIGURED — This system call is not part of the present configuration.

E$TYPE — The task parameter is a token for an object which is not a task.

DISABLE

DISABLE disables an interrupt level.

```
    CALL RQ$DISABLE (level, except$ptr);
```

INPUT PARAMETER

level — A WORD that specifies an interrupt level that is encoded as follows (bit 15 is the high-order bit):

Bits	Value
15-7	Ø
6-4	First digit of the interrupt level (Ø-7)
3	If one, the level is a master level and bits 6-4 specify the entire level number If zero, the level is a slave level and bits 2-Ø specify the second digit
2-Ø	Second digit of the interrupt level (Ø-7), if bit 3 is zero

OUTPUT PARAMETER

except$ptr — A POINTER to a WORD to which the iRMX 86 Operating System will return the condition code generated by this system call. All exceptional conditions must be processed in-line. Control does not pass to an exception handler.

DESCRIPTION

The DISABLE system call disables the specified interrupt level. It has no effect on other levels. To be disabled, a level must have an interrupt handler assigned to it. Otherwise, the Nucleus returns an exception code.

You must not disable the level reserved for the system clock. You determine this level during system configuration (refer to the iRMX 86 CONFIGURATION GUIDE).

DESCRIPTION

The FORCE$DELETE system call deletes objects whose disabling depths are zero or one. If an object has a deletion depth of two or more, the calling task is put to sleep until the deletion depth is decreased to one. At that time, the object is deleted and the task is awakened. If the wrong extension type is specified, FORCE$DELETE issues and E$CONTEXT error and returns without deleting the composite.

CONDITION CODES

E$OK	No exceptional conditions.
E$CONTEXT	The wrong extension type was used in the extension parameter of the FORCE$DELETE system call.
E$EXIST	One or both of the object or extension parameters is not a token for an existing object.
E$MEM	The memory available to the calling task's job is not sufficient to complete this call.
E$NOT$CONFIGURED	This system call is not part of the present configuration.
E$TYPE	The extension parameter is a token for an object that is not an extension object.

GET$EXCEPTION$HANDLER

GET$EXCEPTION$HANDLER returns information about the calling task's exception handler.

```
    CALL RQ$GET$EXCEPTION$HANDLER (exception$info$ptr, except$ptr);
```

OUTPUT PARAMETERS

exception$info$ptr A POINTER to a structure of the following form:

```
STRUCTURE (
    EXCEPTION$HANDLER$OFFSET   WORD,
    EXCEPTION$HANDLER$BASE     WORD,
    EXCEPTION$MODE             BYTE);
```

where, after the call,

- exception$handler$offset contains the offset of the first instruction of the exception handler.
- exception$handler$base contains a base for the segment containing the first instruction of the exception handler. If exception$handler$base and exception$handler$offset are both zero, the calling task's exception handler is the system default exception handler.
- exception$mode contains an encoded indication of the calling task's current exception mode. The value is interpreted as follows:

Value	When to Pass Control to Exception Handler
Ø	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

except$ptr A POINTER to a WORD to which the iRMX 86 Operating System will return the condition code generated by this system call.

DESCRIPTION

The GET$EXCEPTION$HANDLER system call returns both the address of the calling task's exception handler and the current value of the task's exception mode.

GET$TYPE

GET$TYPE returns the encoded type of an object.

```
    type$code = RQ$GET$TYPE (object, except$ptr);
```

INPUT PARAMETER

object — A TOKEN for an object.

OUTPUT PARAMETERS

type$code — A WORD which contains the encoded type of the specified object. The types for iRMX 86 objects are encoded as follows:

Value	Type
1	job
2	task
3	mailbox
4	semaphore
5	region
6	segment
7	extension
1ØØH	composite (user)
1Ø1H	composite (connection)
3ØØH	composite (I/O job)
3Ø1H	composite (logical device)
8ØØØH - ØFFFFH	user-created composites

Users and connections are described in the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL. I/O jobs and logical devices are described in the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL.

except$ptr — A POINTER to a WORD to which the condition code for the call is returned.

DESCRIPTION

The GET$TYPE system call returns the type code for an object.

EXAMPLE

```
/**************************************************************************
 *  This example illustrates how the GET$TYPE system call can be used   *
 *  to return the encoded type of an object.                             *
 **************************************************************************/

    $INCLUDE(:F1:SAMPLE.EXT);           /* Declares all system calls */

    DECLARE TOKEN                       LITERALLY 'SELECTOR';
                                        /* if your PL/M compiler does not
                                           support this variable type,
                                           declare TOKEN a WORD */
    DECLARE type$code                   WORD;
    DECLARE mbx$token                   TOKEN;
    DECLARE calling$tasks$job           LITERALLY 'Ø';
    DECLARE wait$forever                LITERALLY 'ØFFFFH';
    DECLARE object$token                TOKEN;
    DECLARE response                    TOKEN;
    DECLARE status                      WORD;

SAMPLE_PROCEDURE:

    PROCEDURE;

         ●
         ●     Typical PL/M-86 Statements
         ●

/**************************************************************************
 *  In order to invoke the GET$TYPE system call, the calling task must  *
 *  have the token for an object.  In this example, the calling task    *
 *  invokes the LOOKUP$OBJECT system call and then the RECEIVE$MESSAGE  *
 *  system call to receive the token for an object of unknown type      *
 *  (object$token).                                                      *
 **************************************************************************/

    mbx$token = RQ$LOOKUP$OBJECT        (calling$tasks$job,
                                        @(3,'MBX'),
                                        wait$forever,
                                        @status);
         ●
         ●     Typical PL/M-86 Statements
         ●

/**************************************************************************
 *  The RECEIVE$MESSAGE system call returns object$token to the calling *
 *  task after the calling task invoked LOOKUP$OBJECT to receive the    *
 *  token for the mailbox named 'MBX'.  'MBX' had been predesignated    *
 *  as the mailbox another task would use to send an object.            *
 **************************************************************************/

    object$token = RQ$RECEIVE$MESSAGE   (mbx$token,
                                        wait$forever,
                                        @response,
                                        @status);
```

- if unequal to zero, indicates that the calling task is to be the interrupt task that will be invoked by the interrupt handler being set. The priority of the calling task is adjusted by the Nucleus according to the interrupt level being serviced. Table 8-4 lists the levels and the corresponding interrupt task priorities. Be certain that priorities set in this manner do not violate the max$priority attribute of the containing job.

The value of this parameter indicates the number of outstanding SIGNAL$INTERRUPT requests that can exist. When this limit is reached, the associated interrupt level is disabled. The maximum value for this parameter is 255 decimal. Chapter 8 describes this feature in more detail.

interrupt$handler — A POINTER to the first instruction of the interrupt handler. To obtain the proper start address for interrupt handlers written in PL/M-86, place the following instruction before the call to SET$INTERRUPT:

```
interrupt$handler
          = interrupt$ptr (inter);
```

where interrupt$ptr is a PL/M-86 built-in procedure and inter is the name of your interrupt handling procedure.

interrupt$handler$ds — A WORD which specifies the interrupt handler's data segment.

- If not zero, contains the base address of the interrupt handler's data segment. See the description of ENTER$INTERRUPT in this chapter for information concerning the significance of this parameter.

  It is often desirable for an interrupt handler to pass information to the interrupt task that it calls. The following PL/M-86 statements, when included in the interrupt task's code (with the first statement listed here being the first statement in the task's code), will extract the DS register value used by the interrupt task and make it available to the interrupt handler, which in turn can access it by calling ENTER$INTERRUPT:

```
DECLARE BEGIN WORD;   /* A DUMMY VARIABLE */

DECLARE DATA$PTR POINTER;

DECLARE DATA$ADDRESS STRUCTURE (

   OFFSET WORD,

   BASE WORD) AT (@DATA$PTR); /* THIS MAKES
            ACCESSIBLE THE TWO HALVES OF THE
            POINTER DATA$PTR */

DATA$PTR = @BEGIN;  /* PUTS THE WHOLE
            ADDRESS OF THE DATA SEGMENT INTO
            DATA$PTR AND DATA$ADDRESS */

DS$BASE = DATA$ADDRESS.BASE;

CALL RQ$SET$INTERRUPT (...,DS$BASE);
```

- if zero, indicates that the interrupt handler will load its own data segment and may not invoke ENTER$INTERRUPT.

OUTPUT PARAMETER

except$ptr — A POINTER to a WORD to which the iRMX 86 Operating System will return the condition code generated by this system call.

DESCRIPTION

The SET$INTERRUPT system call is used to inform the Nucleus that the specified interrupt handler is to service interrupts which come in at the specified level. In a call to SET$INTERRUPT, a task must indicate whether the interrupt handler will invoke an interrupt task and whether the interrupt handler has its own data segment. If the handler is to invoke an interrupt task, the call to SET$INTERRUPT also specifies the number of outstanding SIGNAL$INTERRUPT requests that the handler can make before the associated interrupt level is disabled. This number generally corresponds to the number of buffers used by the handler and interrupt task. Refer to Chapter 8 for further information.

If there is to be an interrupt task, the calling task is that interrupt task. If there is no interrupt task, SET$INTERRUPT also enables the specified level, which must be disabled at the time of the call.

A$ATTACH$FILE

A$ATTACH$FILE creates a connection to an existing file.

```
    CALL RQ$A$ATTACH$FILE(user, prefix, subpath$ptr, resp$mbox,
                          except$ptr);
```

INPUT PARAMETERS

user — A TOKEN for the user object to be inspected in any access checking that takes place. A zero specifies the default user for the calling task's job. This parameter is ignored when attaching physical or stream files. Access checking does occur for named files.

prefix — A TOKEN for the connection object to be used as the path prefix. A zero specifies the default prefix for the calling task's job.

subpath$ptr — A POINTER to a STRING containing the subpath of the file to be attached. A null string indicates that the new connection is to the file designated by the prefix. The new connection will not be open, regardless of the open mode of the prefix. (This parameter is ignored for physical and stream files.)

OUTPUT PARAMETERS

resp$mbox — A TOKEN for the mailbox into which the Basic I/O System places a token for the result object of the call. This result object is a new file connection if the call succeeds or an I/O result segment otherwise (see Appendix C). To ascertain the type of object returned, use the Nucleus system call GET$TYPE.

If the object received is an I/O result segment, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

DESCRIPTION

A$ATTACH$FILE creates a connection to an existing file. Once the connection is established, it remains in effect until the connection object is deleted, or until the creating job is deleted. Once attached, the file may be opened, closed, read, written, etc., as many times as desired. A$ATTACH$FILE has no effect on the owner ID or the access list for the file.

CONDITION CODES

A$ATTACH$FILE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in Chapter 7 of this manual.

The following list is divided into two parts -- one for sequential codes, and one for concurrent codes.

Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

E$OK — No exceptional conditions.

E$DEV$OFF$LINE — The prefix parameter in this system call refers to a logical connection. One of the following is true of the device associated with the connection:

- It has been physically attached but is now off-line.
- It has never been physically attached. (See Appendix E for a more detailed explanation.)

E$EXIST — One of the following is true:

- One or more of the following parameters is not a token for an existing object:
  - The user parameter
  - The prefix parameter
  - The resp$mbox parameter
- The prefix connection is being deleted.

E$LIMIT

Processing this call would cause one or more of these limits to be exceeded:

- The object limit for this job.

- The number of I/O operations that can be outstanding at one time for the user object specified in the call (255 decimal).

- The number of I/O operations that can be outstanding at one time for the caller's job (255 decimal).

E$MEM

The memory available to the calling task's job is not sufficient to complete the call.

E$NO$PREFIX

The calling task specified a default prefix (prefix argument equals zero), but no default prefix can be found because of one of the following reasons:

- When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default prefix.

- The job's directory can have entries but a default prefix is not cataloged there.

E$NO$USER

If the user parameter in this call is not zero, the parameter is not a token for a user object.

If the user parameter is zero, it specifies a default user. But no default user can be found because of one of the following reasons:

- When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default user.

- The job's directory can have entries but a default user is not cataloged there.

- The object that is cataloged with the name R?IOUSER is not a user object. The name R?IOUSER should be treated as a reserved word.

E$NOT$CONFIGURED

This system call is not part of the present configuration.

E$TYPE

One of more of the following conditions caused this exception:

- The prefix parameter is a token for an object that is not of the correct type. It must be either a connection object or a logical device object. (Logical device objects are created by the Extended I/O System.)
- The resp$mbox parameter in the call is a token for an object that is not a mailbox.

Concurrent Condition Codes

The Basic I/O System can return the following condition codes in an I/O result segment at the mailbox specified by resp$mbox. After examining the segment, you should delete it.

E$OK

No exceptional conditions.

E$DEV$DETATCHING

The file specified is on a device that the system is detaching.

E$FNEXIST

A file in the specified path, or the target file itself, does not exist or is marked for deletion.

E$FTYPE

The string pointed to by the subpath$ptr parameter contains a filename that should be the name of a directory, but is not. (Except for the last file, each file in a path must be a named directory.)

E$INVALID$FNODE

The fnode for the specified file is invalid. The file cannot be accessed; you should delete it.

E$IO

An I/O error occurred, which might have prevented the operation from completing. Examine the unit$status field of the I/O result segment for more information.

E$IO$MEM

The memory available to the Basic I/O System job is not sufficient to complete the call.

A$CHANGE$ACCESS

A$CHANGE$ACCESS changes the access rights to a named data or directory file.

```
    CALL RQ$A$CHANGE$ACCESS(user, prefix, subpath$ptr, id, access,
                            resp$mbox, except$ptr);
```

INPUT PARAMETERS

user — A TOKEN for the user object to be inspected in access checking. A value of zero specifies the default user for the calling task's job.

prefix — A TOKEN for the connection object to be used as the path prefix. A zero specifies the default prefix for the calling task's job.

subpath$ptr — A POINTER to a STRING giving the subpath of the file whose access is to be changed. A null string indicates that the prefix itself designates the desired file.

id — A WORD containing the ID number of the user whose access is to be changed. If this ID does not already exist in the ID-access mask list, it is added. This list may contain a total of three ID-access pairs.

access — A BYTE mask giving the new access rights for the ID. For each bit, a one grants access, and a zero denies it. (Bit Ø is the low-order bit.) For a named data file, the possible bit settings are:

Bit	Meaning
Ø	Delete
1	Read
2	Append
3	Update
4-7	Reserved (set to Ø)

For a named directory file, the possible bit settings are:

Bit	Meaning
Ø	Delete
1	Display
2	Add Entry
3	Change Entry
4-7	Reserved (set to Ø)

If zero is specified for the access parameter (that is, no access), the ID specified in the id parameter is deleted from the file's ID-access list.

OUTPUT PARAMETERS

resp$mbox

A TOKEN for the mailbox that receives an I/O result segment indicating the result of the call (see Appendix C). A value of zero means that you do not want to receive an I/O result segment.

If it receives an I/O result segment, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr

A POINTER to a WORD where the sequential condition code will be returned.

DESCRIPTION

A$CHANGE$ACCESS system call applies to named files only. This call has no effect on existing connections to the file. It is called to change the access rights to a named data or directory file. Depending on the contents of the "id" and "access" parameters specified in the system call, users may be added to or deleted from the file's ID-access mask list, or the access privileges granted to a particular user may be changed.

NOTE

The caller must be the owner of the file or must have change entry access to the file's parent directory. However, if the owner is "WORLD", that is, ØFFFFH, then any task may change the access mask of the file.

granularity

A WORD giving the granularity of the file being created. This is the size (in bytes) of each logical block to be allocated to the file. The value specified in this parameter is rounded up, if necessary, to a multiple of the volume granularity. Note that a contiguous file can become noncontiguous when it is extended.

The granularity parameter can have the following values:

Ø	Same as volume granularity
FFFF	The file must be contiguous
Other	Number of bytes per allocation

When a contiguous file is extended, space is allocated in volume-granularity units. If "Other" is specified, a multiple of 1Ø24 bytes is recommended.

This parameter is ignored for physical and stream files.

size

A DWORD giving the number of bytes initially reserved for the file. For stream files, this value must equal zero. For physical files, this parameter is ignored.

must$create

A BYTE whose value (ØFFH for TRUE or Ø for FALSE) determines the handling of input paths designating an existing file (see following DESCRIPTION). This parameter applies only to named files.

OUTPUT PARAMETERS

resp$mbox

A TOKEN for the mailbox that receives the result object of this call. This result object is a new file connection if the call succeeded or an I/O result segment otherwise (see Appendix C). To ascertain the type of object returned, use the Nucleus system call GET$TYPE.

If the object received is an I/O result segment, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr

A POINTER to a WORD where the sequential condition code will be returned.

DESCRIPTION

The A$CREATE$FILE system call creates a physical, stream, or named data file and returns a token for the new file connection. If a named file designated by the prefix and subpath parameters already exists, one of the following occurs:

- Error: If the "must$create" parameter is TRUE (ØFFH), an error condition code (E$FEXIST) is returned.

- Truncate File: If the "must$create" parameter is FALSE (Ø) and the path designates an existing data file, a new connection to that file is returned (that is, A$CREATE$FILE acts like A$ATTACH$FILE). In this case, the file is truncated or expanded according to the "size" parameter, so data in the file might be lost. As in the case of A$ATTACH$FILE, the file's owner ID and access list are unchanged.

- Temporary File Created: If the "must$create" parameter is FALSE (Ø), and the path designates an existing directory file or device, an unnamed temporary file is created on the corresponding device. This file is deleted automatically when the last connection to it is deleted. Because this file is created without a path, it can be accessed only through a connection.

  Any task can create a temporary file by referring to any directory. This is true because temporary files are not listed as ordinary entries in the directory, so no add-entry access is required.

Many of the parameters specified in the A$CREATE$FILE call do not apply to physical and stream files. In these cases, the parameter is ignored.

NOTE

The caller must have add-entry access to the parent directory of the new named file.

CONDITION CODES

A$CREATE$FILE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent exception code. A complete explanation of sequential and concurrent parts of system calls is in Chapter 7 of this manual.

The following list is divided into two parts -- one for sequential codes, and one for concurrent codes.

A$SPECIAL

A$SPECIAL enables tasks to perform a variety of special functions.

```
CALL RQ$A$SPECIAL(connection, spec$func, ioparm$ptr, resp$mbox,
                  except$ptr);
```

INPUT PARAMETERS

connection — A TOKEN for a connection to the file or device for which the special function is to be performed.

spec$func — An encoded WORD that, with the connection argument, specifies the function being requested. The functions are described under the heading DESCRIPTION and are summarized as follows:

File driver	Spec$func	
for connection	value	Function
Physical	Ø	Format track
Stream	Ø	Query
Stream	1	Satisfy
Physical or Named	2	Notify
Physical	3	Get disk/tape data
Physical	4	Get terminal data
Physical	5	Set terminal data
Physical	6	Set signal
Physical	7	Rewind tape
Physical	8	Read tape file mark
Physical	9	Write tape file mark
Physical	1Ø	Retention tape
	11-32767	Reserved for other Intel products

ioparm$ptr — A POINTER to a parameter block. The contents of the parameter block depends upon the requirements of the special function being requested and are described fully under the heading DESCRIPTION. Enter a zero value if the special function you request does not require a parameter block.

OUTPUT PARAMETERS

resp$mbox — A TOKEN for the mailbox that receives an I/O result segment indicating the result of the call (see Appendix C). A value of zero means that you do not want to receive an I/O result segment.

If it receives an I/O result segment, the calling task should call DELETE$SEGMENT to delete the segment.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

DESCRIPTION

The A$SPECIAL system call enables tasks to perform a variety of special functions.

Tasks define their requests by means of the spec$func and ioparm$ptr parameters. Spec$func is a code which, when combined with the file driver associated with the connection argument, specifies the function the Basic I/O System is to perform. When more information is needed to define a request, ioparm$ptr points to a parameter block containing the additional data. Descriptions of the available functions follow.

Formatting a Track. This function applies to physical files only. To format a track on a mass storage device, call A$SPECIAL with an open file connection, with spec$func equal to Ø, and with ioparm$ptr pointing to a structure of the form:

```
DECLARE format$track STRUCTURE(
   track$number    WORD,
   interleave      WORD,
   track$offset    WORD,
   fill$char       WORD);
```

In this structure, the fields are defined as follows:

track$number — The number of the track to be formatted. Acceptable values are Ø to one less than the number of tracks on the volume. Other values cause an E$SPACE exceptional condition. When formatting a RAM-disk or a tape, you must place a zero value in this field.

interleave — The interleave factor for the track. (That is, the number of physical sectors to advance when locating the next logical sector.) An interleave factor of zero or one skips no physical sectors between logical sectors. If the specified

Bits Value and Meaning

2 Output medium (corresponds to OSC characters T:H).

Ø = Video display terminal (VDT).

1 = Printed (Hard copy).

3 Modem indicator (corresponds to OSC characters T:M).

Ø = Not used with a modem.

1 = Used with a modem.

4-5 Input parity control (corresponds to OSC characters T:R). The parity bit (bit 7) of each input byte can be used in a variety of ways. A byte has even parity if the sum of its bits is an even number. Otherwise, the byte has odd parity.

Ø = Always set parity bit to Ø.

1 = Never alter the parity bit.

2 = Even parity is expected on input. Use the parity bit to indicate the presence (1) or absence (Ø) of an error on input. That is, set the parity bit to Ø unless the received byte has odd parity or there is some other error, such as (a) the received stop bit has a value of Ø (framing error) or (b) the previous character received has not yet been fully processed (overrun error.)

3 = Odd parity is expected in input. Use the parity bit to indicate the presence (1) or absence (Ø) of an error on input. That is, set the parity bit to Ø unless the received byte has even parity or there is some other error, such as (a) the received stop bit has a value of Ø (framing error) or (b) the previous character received has not yet been fully processed (overrun error.)

Bits Value and Meaning

6-8 Output parity control (corresponds to OSC characters T:W). The parity bit (bit 7) of each output byte can be used in a variety of ways. A byte has even parity if the sum of its bits is an even number. Otherwise, the byte has odd parity.

Ø = Always set parity bit to Ø.

1 = Always set parity bit to 1.

2 = Set parity bit to give the byte even parity.

3 = Set parity bit to give the byte odd parity.

4-7 = Do not alter parity bit.

9 Translation control (corresponds to OSC characters T:T). Translation refers to the ability to define certain control characters so that whenever these characters are entered at or written to a terminal, certain actions, usually cursor movements, take place automatically. Translation is described in Appendix F of this manual.

Ø = Do not enable translation.

1 = Enable translation.

1Ø Terminal axes sequence control (corresponds to OSC characters T:F). This specifies the order in which Cartesian-like coordinates of elements on a terminal's screen are to be listed or entered.

Ø = List or enter the horizontal coordinate first.

1 = List or enter the vertical coordinate first.

11 Horizontal axis orientation control (corresponds to OSC characters T:F). This specifies whether the coordinates on the terminal's horizontal axis increase or decrease as you move from left to right across the screen.

Bits **Value and Meaning**

Ø = Coordinates increase from left to right.

1 = Coordinates decrease from left to right.

12 Vertical axis orientation control (corresponds to OSC characters T:F). This specifies whether the coordinates on the terminal's vertical axis increase or decrease as you move from top to bottom across the screen.

Ø = Coordinates increase from top to bottom.

1 = Coordinates decrease from top to bottom.

13-15 Reserved bits. For future compatibility, set to Ø.

NOTE

If bits 4-5 contain 2 or 3, and bits 6-8 also contain 2 or 3, then they must both contain the same value. That is, they must both reflect the same parity convention (even or odd).

in$baud$rate

The input baud rate indicator (corresponds to OSC characters T:I). If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The word is encoded as follows:

Ø = Invalid.

1 = Perform an automatic baud rate search.

Other = Actual input baud rate, such as 96ØØ.

out$baud$rate

The output baud rate indicator (corresponds to OSC characters T:O). If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The word is encoded as follows:

Ø = Leave field set to previous value

1 = Use the input baud rate for output.

Other = Actual output baud rate, such as 96ØØ.

Most applications require the input and output baud rates to be equal. In such cases, use in$baud$rate to set the baud rate and specify a one for out$baud$rate.

scroll$lines

An operator at a terminal can enter a control character (default is Control-W) when he/she is ready for data to appear on the terminal's display screen. The scroll$lines value (corresponding to OSC characters T:S) specifies the maximum number of lines that are to be sent to the terminal each time the operator enters the control character. If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value.

x$y$size

The low-order byte of this word specifies the number of character positions on each line of the terminal's screen (and corresponds to OSC characters T:X). The high-order byte specifies the number of lines on the terminal's screen (and corresponds to OSC characters T:Y).

x$y$offset

The low-order byte of this word specifies the value that starts the numbering sequence of both the X and Y axes (and corresponds to OSC characters T:U). The high-order byte specifies the value to which the numbering of the axes must "fall back" after reaching 127 (and corresponds to OSC characters T:V).

The remaining fields apply only for intelligent communications boards (such as the iSBC 544 board) that maintain their own input and output buffers separately from the ones managed by the Basic I/O System's Terminal Support Code. If you aren't sure whether you can set these fields, invoke A$SPECIAL with function code 4 to get the terminal attributes. If bit 15 of the flow$control field (the next one described) is set, your board is a buffered device and you can set the following fields. (If your board is not a buffered device, setting any of the following fields will cause the terminal support code to return an E$PARAM Condition Code.)

flow$control

Specifies whether the communications board sends flow control characters (selected by the fc$on$char and fc$off$char fields, but usually XON and XOFF) to turn input on and off (corresponds to the OSC characters T:G). The low-order bit (bit Ø) controls this option, as follows:

Ø	Disable flow control.
1	Enable flow control.

```
    DELCLARE   read$file$mark    STRUCTURE(
       search           BYTE);
```

Where:

search — A value indicating the direction of the search, as follows:

ØØ Search forward

ØFFH Search backward (for start/stop drives only)

When your task issues the A$SPECIAL system call with spec$func set to 9, the tape drive writes a file mark at the current position on the tape. This function also terminates tape write operations.

When your task issues the A$SPECIAL system call with spec$func set to 1Ø, the tape drive fast-forwards the tape to the end and then rewinds it to the load point.

CONDITION CODES

A$SPECIAL return condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a <u>sequential</u> condition code. A code returned as a result of asynchronous processing is a <u>concurrent</u> condition code. A complete explanation of sequential and concurrent parts of system calls is in Chapter 7 of this manual.

The following list is divided into two parts -- one for sequential codes, and one for concurrent codes.

Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

E$OK — No exceptional conditions.

E$BUFFERED$CONN — The connection parameter is a connection produced by the Extended I/O System. You cannot use it with Basic I/O System calls.

E$EXIST — At least one of the following is true:

- One or more of the following parameters or fields is not a token for an existing object:

  - The connection parameter
  - The resp$mbox parameter
  - The mailbox field in the notify structure. (Spec$func = 2.)
  - The object field in the notify structure. (Spec$func = 2.)
  - The semaphore field in the signal$pair structure. (Spec$func = 6.)

- The connection is being deleted.

E$IFDR — The function requested (spec$func) is not valid for the type of file specified by the connection parameter.

E$LIMIT — The calling task's job has already reached its object limit.

E$MEM — The memory available to the calling task's job is not sufficient to complete the call.

E$NOT$CONFIGURED — This system call is not part of the present configuration.

E$PARAM — At least one of the following is true:

- The spec$func parameter was 6, and the character field was greater than 1FH.
- The spec$func parameter was greater than 1Ø.
- One or more of the fields related to buffered devices (high$water$mark, low$water$mark, fc$on$char, fc$off$char) was set while bit 15 of the flow$control field was reset to zero (specifying an unbuffered device).

E$SUPPORT — The specified connection was not created by this job.

E$TYPE — One or more of the following parameters or fields is a token for an existing object of the wrong type:

- The connection parameter.
- The resp$mbox parameter.
- The mailbox field of the notify structure. (Spec$func = 2.)
- The semaphore field of the signal$pair structure. (Spec$func = 6.)

Concurrent Condition Codes

The Basic I/O System can return the following condition codes in an I/O result segment at the mailbox specified by resp$mbox. After examining the segment, you should delete it.

E$OK — No exceptional conditions.

E$CONN$NOT$OPEN — The specified connection is not open. This applies only to stream and physical files.

E$FLUSHING — The specified connection was closed before the function could be completed.

E$IDDR

The specified function is not supported by the device containing the file.

E$IO — An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the I/O result segment for more information.

E$NOT$DEVICE$CONN — The function code is 'notify', but the specified connection is not a device connection. This applies only to named and physical files.

E$PARAM — The spec$func parameter was 5 while bits Ø-1 of the connection$flags field was equal to Ø.

E$SPACE — One of the following is true:

- This call attempted to format a track of a physical file that is beyond the end of the volume.
- This call attempted to format a track of a RAM disk other than track Ø.

E$STREAM$SPECIAL One of the following is true:

- This is a "query" request, but another query is already queued. This applies only to stream files.

- This is a "satisfy" request, but either a query request is queued, or no requests are queued. This applies only to stream files. (See Artificially Satisfying a Stream File I/O Request in the DESCRIPTION.)

E$TYPE	At least one of the following is true: ● The connection parameter is a token for an object that is not a connection. ● The resp$mbox parameter is a token for an object that is not a mailbox.

Concurrent Condition Codes

The Basic I/O System can return the following condition codes in an I/O result segment at the mailbox specified by resp$mbox. After examining the segment, you should delete it.

E$OK	No exceptional conditions.
E$CONN$NOT$OPEN	The specified file is not open for writing or updating.
E$IO	An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the I/O result segment for more information.

A$UPDATE

A$UPDATE updates a device by writing all partial sectors that remain in the Basic I/O System's buffers after the most recent A$WRITE call.

```
    CALL RQ$A$UPDATE(connection, resp$mbox, except$ptr);
```

INPUT PARAMETERS

connection — A TOKEN for a file or device connection. A$UPDATE updates all files on the device.

resp$mbox — A TOKEN for the mailbox that receives an I/O result segment indicating the result of the call (see Appendix C). A value of zero means that you do not want to receive an I/O result segment.

If it receives an I/O result segment, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

OUTPUT PARAMETER

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

DESCRIPTION

When the I/O System performs an A$WRITE operation, it writes only entire sectors. If part of a sector remains to be written, the I/O System, unless requested to finish the writing operation (that is, to "update the file"), leaves the data for a partial sector in an output buffer. The next time A$WRITE is called on behalf of that file, the I/O System combines the leftover data in the buffer with the data in the new request and again begins writing entire sectors.

The A$UPDATE system call forces the Basic I/O System to finish the writing operation for a device; that is, it writes all partial buffers pertaining to files on a particular device. This ensures that files on removable volumes (such as diskettes) are updated before the operator removes the volume. However, the A$UPDATE system call has no effect on buffers that the Extended I/O System manages.

CONDITION CODES

E$OK — No exceptional conditions.

E$EXIST — The user parameter is not a token for an existing object.

E$NOT$CONFIGURED — This system call is not part of the present configuration.

E$PARAM — The length field contains a zero value.

E$TYPE — The user parameter is a token for an object of the wrong type.

SET$DEFAULT$PREFIX

SET$DEFAULT$PREFIX sets the default prefix for an existing job.

```
    CALL RQ$SET$DEFAULT$PREFIX(job, prefix, except$ptr);
```

INPUT PARAMETERS

job	A TOKEN for the job whose default prefix is to be set. A zero specifies the current job.
prefix	A TOKEN for the connection that is to become the default prefix.

OUTPUT PARAMETERS

except$ptr	A POINTER to a WORD where the condition code will be returned.

DESCRIPTION

The SET$DEFAULT$PREFIX system call sets the default prefix for an existing job. It does this by cataloging the connection (supplied as the prefix parameter) in the object directory of the job (supplied as the job parameter). The Basic I/O System catalogs the prefix under the name "$". If an object is already cataloged under the name "$", the Basic I/O System uncatalogs that object before cataloging the new prefix.

CONDITION CODES

E$OK — No exceptional conditions.

E$CONTEXT — When this job was created, a size of zero was specified for the object directory, so a default prefix cannot be cataloged

E$EXIST — One or more of the following parameters is not a token for an existing object:

- The job parameter

- The prefix parameter

create a segment every time an I/O result segment is needed. This provides a significant advantage because A$READ, A$WRITE, and A$SEEK are typically the most commonly invoked Basic I/O System calls.

CONDITION CODES

E$OK
No exceptional conditions.

E$EXIST
At least one of the following is true:

- The connection parameter or the resp$mbox parameter (or both) did not contain a token for an existing object.
- The specified connection or response mailbox (or both) was deleted.
- The token returned to the specified mailbox was for an object that had been deleted.

E$IO$HARD
A hard I/O error occurred. Another retry is probably useless.

E$IO$MODE
At least one of the following is true:

- A tape drive attempted to perform a read operation before the previous write operation completed.
- A tape drive attempted to perform a write operation before the previous read operation completed.

E$IO$NO$DATA
A tape drive attempted to read the next record, but it found no data.

E$IO$OPRINT
The device was off-line. Operator intervention is required.

E$IO$SOFT
A soft I/O error occurred. The Basic I/O System tried to perform the operation a number of times (the number is configurable for each device). All attempts failed. If the configurable value specifying the number of retries is a reasonable value (for example, 9), another retry probably won't be successful either.

E$IO$UNCLASS
An unknown type of I/O error occurred.

E$IO$WRPROT
The asynchronous operation was A$WRITE and the volume was write-protected.

E$NOT$CONFIGURED
This system call is not part of the present configuration.

E$TIME

One of the following is true:

- The calling task was not willing to wait, and there was no I/O result segment at the specified mailbox.

- The specified waiting period elapsed before the response mailbox received an I/O result segment.

E$TYPE

At least one of the following is true:

- The connection parameter is a token for an object that is not a file connection.

- The resp$mbox parameter is a token for an object that is not a mailbox.

- The object received at the response mailbox is not a segment or is a segment that is not an I/O result segment.

***

- If equal to zero, specifies that the new job's initial task is to have a priority equal to the the maximum priority of the initial job of the Extended I/O System. For more information about the initial job of the Extended I/O System, refer to the chapter of the iRMX CONFIGURATION GUIDE relating to the Extended I/O System.

- If not equal to zero, contains the priority of the initial task of the new job. If this priority is higher than (numerically less than) the maximum priority of the initial job of the Extended I/O System, an E$PARAM error occurs.

start$address — A POINTER to the first instruction of the code segment for the new job's initial task. This code segment can be, but is not required to be, an iRMX 86 segment.

data$seg — A WORD which,

- if zero, indicates one of two things. Either the new job's initial task uses no data segment, or it creates one for itself. Tasks can create their own data segments only under special circumstances. To find out more about the circumstances, refer to the iRMX 86 CONFIGURATION GUIDE.

- if not zero, contains the base address of the data segment of the new job's initial task. This data segment can be, but is not required to be, an iRMX 86 segment.

stack$ptr — A POINTER which,

- if the base portion is zero, specifies that the Nucleus should allocate a stack for the new job's initial task. The length of the allocated stack is determined by the stack$size parameter of this system call. Be aware that this stack is not an iRMX 86 segment.

- if the base portion is not equal to zero, points to the base of the stack for the new job's initial task. Because the Nucleus does not allocate this stack, you must allocate it during the configuration process, or your application code must allocate it while the system is running.

stack$size — A WORD containing the size, in bytes, of the stack for the new job's initial task. If you specify less than 200, the Extended I/O System will increase the size to 200. For information regarding the amount of stack to allocate, refer to the chapter of the iRMX 86 PROGRAMMING TECHNIQUES manual that discusses stack sizes.

If you are allocating the stack during configuration, or if the application code is allocating the stack while the system is running, the value of this parameter will be the precise amount of stack that the system can use. However, if the Nucleus is allocating the stack for you, it might allocate as many as 15 additional bytes in order to make the stack occupy whole 16-byte paragraphs.

task$flags — A WORD in which all bits except the two low-order bits are set to zero.

Bit Zero: Use the low-order bit (bit Ø) to tell the Operating System whether the new job's initial task uses floating-point instructions. A value of 1 indicates the presence of floating-point instructions, while a zero indicates the absence of floating-point instructions.

Bit One: Bit 1 indicates whether the initial task in the job should run immediately, or whether it should wait until a START$IO$JOB system call is issued to start it. Set bit 1 to zero if the task is to be made ready to run; set bit 1 to one if the task is to wait until the START$IO$JOB call is issued.

msg$mbox — A TOKEN for a mailbox. When a task exits (by invoking EXIT$IO$JOB), the Extended I/O System sends a message to this mailbox. If you desire no such message, assign msg$mbox a value of zero.

The format of the message is as follows:

```
DECLARE message    STRUCTURE(
    termination$code  WORD,
    user$fault$code   WORD,
    job$token         WORD,
    return$data$len   BYTE,
    return$data(*)    BYTE)
```

where:

termination$code — A WORD that indicates why an I/O job terminated, as follows:

CODE	MEANING
Ø	Some task within the job -- the terminating task -- invoked the EXIT$IO$JOB system call, and indicated with this code that no problem caused the termination. The job has not yet been deleted, and some of its tasks might still be ready.

APPENDIX C. CONDITION CODES

The iRMX 86 Extended I/O System uses condition codes to inform your tasks of any problems that occur during the execution of a system call. If no problems occur and the system call runs to completion, the Extended I/O System returns an E$OK condition code. Otherwise, the Extended I/O System returns an exceptional condition code.

The meaning of a specific exceptional condition code depends upon the system call that returns the code. For this reason, this appendix does not list any interpretations.

This appendix provides you with the numeric value associated with each condition code that the Extended I/O System can return. To use the exception code values in a symbolic manner, you can assign (using the PL/M-86 "literally" statement) a meaningful name to each of the codes.

The following list correlates the name of the condition code (as described in Chapter 7 of this manual) to the value that the Extended I/O System actually returns. The list is divided into three parts; one for the normal condition code, one for exception codes that indicate a programming error, and one for exception codes that indicate an environmental condition.

NORMAL CONDITION CODE

NAME OF CONDITION	DECIMAL	HEXADECIMAL
E$OK	Ø	ØH

PROGRAMMING ERRORS

NAME OF CONDITION	DECIMAL	HEXADECIMAL
E$ZERO$DIVIDE	32768	8ØØØH
E$OVERFLOW	32769	8ØØ1H
E$TYPE	3277Ø	8ØØ2H
E$PARAM	32772	8ØØ4H
E$NOT$SUPPORTED	32773	8ØØ5H
E$NOUSER	328Ø1	8Ø21H
E$NO$PREFIX	328Ø2	8Ø22H
E$NOT$LOG$NAME	32832	8Ø4ØH
E$NOT$DEVICE	32833	8Ø41H
E$NOT$CONNECTION	32834	8Ø42H

ENVIRONMENTAL CONDITIONS

NAME OF CONDITION	DECIMAL	HEXADECIMAL
E$TIME	1	1H
E$MEM	2	2H
E$LIMIT	4	4H
E$CONTEXT	5	5H
E$EXIST	6	6H
E$NOT$CONFIGURED	8	8H
E$FEXIST	32	2ØH
E$FNEXIST	33	21H
E$DEVFD	34	22H
E$SUPPORT	35	23H
E$FACCESS	38	26H
E$FTYPE	39	27H
E$SHARE	4Ø	28H
E$SPACE	41	29H
E$IDDR	42	2AH
E$FLUSHING	44	2CH
E$ILLVOL	45	2DH
E$IFDR	47	2FH
E$FRAGMENTATION	48	3ØH
E$DIR$NOT$EMPTY	49	31H
E$NOT$FILE$CONN	5Ø	32H
E$CONN$NOT$OPEN	52	34H
E$CONN$OPEN	53	35H
E$ALREADY$ATTACHED	56	38H
E$DEV$DETACHING	57	39H
E$NOT$SAME$DEV	58	3AH
E$ILLOGICAL$RENAME	59	3BH
E$STREAM$SPECIAL	6Ø	3CH
E$INVALID$FNODE	61	3DH
E$PATHNAME$SYNTAX	62	3EH
E$FNODE$LIMIT	63	3FH
E$LOG$NAME$SYNTAX	64	4ØH
E$IOMEM	66	42H
E$MEDIA	68	44H
E$LOG$NAME$NEXIST	69	45H
E$NOT$OWNER	7Ø	46H
E$IO$JOB	71	47H
E$IO$UNCLASS	8Ø	5ØH
E$IO$SOFT	81	51H
E$IO$HARD	82	52H
E$IO$OPRINT	83	53H
E$IO$WRPROT	84	54H
E$IO$NO$DATA	85	55H
E$IO$MODE	86	56H

iRMX™ 86 Release 6.Ø Change Package: Update 3

Change Pages for:

iRMX™ 86 Programmer's Reference Manual, Part II (146196-ØØ1)

CONDITION CODES

The A$LOAD system call can return condition codes at two different times. Codes returned to the calling task immediately after invocation of the system call are sequential condition codes. Codes returned after the concurrent part of the system call has finished running are concurrent condition codes. The following list is divided into two parts -- one for sequential codes and one for concurrent codes:

Sequential Condition Codes

The Loader can return any of the following condition codes to the WORD pointed to by the except$ptr parameter of this system call.

E$OK — No exceptional conditions.

E$BAD$HEADER — The target file does not begin with a valid header record for a loadable object module. Possibly the file is a directory.

E$CHECKSUM — The header record of the target file contains a checksum error.

E$CONN$NOT$OPEN — The Loader opened the connection but some other task closed the connection before the loading operation was begun.

E$CONN$OPEN — The calling task specified a connection that was already open.

E$EXIST — At least one of the following is true:

- The connection parameter is not a token for an existing object.
- The msg$mbox parameter did not refer to an existing object.
- The mailbox specified in the response$mbox parameter was deleted before the loading operation was completed.

E$FACCESS — The specified connection did not have "read" access to the file.

E$FLUSHING — The device containing the target file is being detached.

E$IO$HARD — A hard I/O error occurred. This means that another try is probably useless.

E$IO$OPRINT — The device containing the target file was off-line. Operator intervention is required.

E$IO$SOFT A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.

E$IO$UNCLASS An unknown type of I/O error occurred.

E$IO$WRPROT The volume is write-protected.

E$LIMIT At least one of the following is true:

- The calling task's job has already reached its object limit.
- Either the calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations.

E$LOADER$SUPPORT To load the target file requires capabilities not configured into the Loader. For example, it might be attempting to load PIC when configured to load only absolute code.

E$MEM The memory available to the calling task's job or the Basic I/O System is not sufficient to complete the call.

E$NOT$FILE$CONN The calling task specified a connection to a device rather than to a named file.

E$SHARE The calling task tried to open a connection to a file already being used by some other task, and the file's sharing attribute is not compatible with the open request.

E$SUPPORT The specified connection was not created by the calling task's job.

E$TYPE The connection parameter is a token for an object that is not a connection.

Concurrent Condition Codes

After the Loader attempts the loading operation, it returns a condition code in the except$code field of the Loader Result Segment. The Loader can return the following condition codes in this manner.

E$OK No exceptional conditions.

E$BAD$GROUP The target file contains an invalid group definition record.

E$BAD$SEGMENT The target file contains an invalid segment definition record.

E$CHECKSUM — At least one record of the target file contains a checksum error.

E$EOF — The call encountered an unexpected end-of-file.

E$EXIST — The device containing the file to be loaded was detached before the loading operation was completed.

E$FIXUP — The target file contains an invalid fixup record.

E$FLUSHING — The device containing the target file is being detached.

E$IO$HARD — A hard I/O error occurred. This means that another try is probably useless.

E$IO$OPRINT — The device containing the target file was off-line. Operator intervention is required.

E$IO$SOFT — A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.

E$IO$UNCLASS — An unknown type of I/O error occurred.

E$IO$WRPROT — The volume is write-protected.

E$LIMIT — The calling task's job has already reached its object limit.

E$NO$LOADER$MEM — The memory pool of the calling task does not currently have a block of memory large enough to allow the Loader to run.

E$NO$MEM — The Loader attempted to load PIC or LTL groups or segments, but the memory pool of the calling task's job does not currently contain a block of memory large enough to accommodate these groups or segments.

E$NOSTART — The target file does not specify the entry point for the program being loaded.

E$PARAM — The target file has a stack smaller than 16 bytes.

E$REC$FORMAT — At least one record in the target file contains a format error.

E$REC$LENGTH — The target file contains a record longer than the Loader's internal buffer. The Loader's buffer length is specified during the configuration of the Loader. See Chapter 3 and the iRMX 86 CONFIGURATION GUIDE for information about configuring the Loader.

E$REC$TYPE At least one of the following is true:

- At least one record in the target file is of a type that the Loader cannot process.
- The Loader encountered records in a sequence that it cannot process.

E$SEG$BOUNDS The Loader created a segment into which to load code. One of the data records specified a load address outside of that segment.

CONDITION CODES

This system call can return condition codes at two different times. Codes returned to the calling task immediately after the invocation of the system call are considered sequential condition codes. Codes returned after the concurrent part of the system call has finished running are considered concurrent condition codes. The following list is divided into two parts -- one for sequential codes and one for concurrent codes.

Sequential Condition Codes

The Loader returns one of the following condition codes to the WORD pointed to by the except$ptr parameter:

E$OK — No exceptional conditions.

E$BAD$HEADER — The target file does not begin with a valid header record for a loadable object module. Possibly the file is a directory.

E$CHECKSUM — The header record of the target file contains a checksum error.

E$CONN$NOT$OPEN — The Loader opened the connection, but some other task closed the connection before the loading operation was begun.

E$CONN$OPEN — The specified connection was already open.

E$CONTEXT — The calling task's job is not an I/O job.

E$EXIST — At least one of the following is true:

- The connection parameter is not a token for an existing object.
- The calling task's job has no global job.
- The msg$mbox parameter does not refer to an existing object.

E$FACCESS — The specified connection does not have "read" access to the file.

E$FLUSHING — The device containing the target file is being detached.

E$IO$HARD — A hard I/O error occurred. This means that another try is probably useless.

E$IO$OPRINT — The device containing the target file is off-line. Operator intervention is required.

E$IO$SOFT A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.

E$IO$UNCLASS An unknown type of I/O error occurred.

E$IO$WRPROT The volume is write-protected.

E$JOB$PARAM The pool$upper$bound parameter is both non-zero and smaller than the pool$lower$bound parameter.

E$JOB$SIZE The pool$upper$bound parameter is non-Ø and too small for the target file.

E$LOADER$SUPPORT The target file requires capabilities not configured into the Loader. For example, the loader might be attempting to load PIC code when configured to load only absolute code.

E$MEM The memory available to the calling task's job or the Basic I/O System is not sufficient to complete the call.

E$NO$LOADER$MEM The memory pool of the newly created I/O job does not currently have a block of memory large enough to allow the Loader to run.

E$NOT$CONFIGURED This system call is not part of the present configuration.

E$NOT$FILE$CONN The specified connection is to a device rather than to a named file.

E$NOUSER The calling task's job does not have a default user, or the object cataloged under the logical name R?IOUSER is not a user object.

E$PARAM The value of the except$mode field within the except$handler structure lies outside the range Ø through 3.

E$SHARE The calling task tried to open a connection to a file already being used by some other task, and the file's sharing attribute is not compatible with the open request.

E$SUPPORT The specified connection was not created in this job.

E$TIME The calling task's job is not an I/O job.

E$TYPE The connection parameter is a token for an object that is not a connection.

E$EXIST — At least one of the following is true:

- The msg$mbox parameter is not a token for an existing object.
- The calling task's job has no global job.
- The device containing the target file was detached.

E$FACCESS — The default user object for the new I/O job does not have "read" access to the specified file.

E$FIXUP — The target file contains an invalid fixup record.

E$FNEXIST — The specified target file, or some file in the specified path, does not exist or is marked for deletion.

E$FLUSHING — The device containing the target file is being detached.

E$INVALID$FNODE — The fnode for the specified file is invalid, so the file must be deleted.

E$IO$HARD — A hard I/O error occurred. This means that another try is probably useless.

E$IO$JOB — The calling task's job is not an I/O job.

E$IO$OPRINT — The device containing the target file is off-line. Operator intervention is required.

E$IO$SOFT — A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.

E$IO$UNCLASS — An unknown type of I/O error occurred.

E$IO$WRPROT — The volume is write-protected.

E$JOB$PARAM — The pool$upper$bound parameter is nonzero and smaller than the pool$lower$bound parameter.

E$JOB$SIZE — The pool$upper$bound parameter is nonzero and too small for the target file.

E$LIMIT — At least one of the following is true:

- The task$priority parameter is higher (numerically lower) than the newly-created I/O job's maximum priority. This maximum priority is specified during the configuration of the Extended I/O System (if the job is a descendant of the Extended I/O System) or of the Human Interface (if the job is a descendant of the Human Interface).

- Either the newly created I/O job or its default user object is already involved in 255 (decimal) I/O operations.

E$LOADER$SUPPORT The target file requires capabilities not configured into the Loader. For example, it might be attempting to load PIC when configured to load only absolute code.

E$MEM The memory available to the calling task's job is not sufficient to complete the call.

E$NO$LOADER$MEM The memory pool of the newly created I/O job does not currently have a block of memory large enough to allow the Loader to run.

E$NOMEM The target file contains either PIC segments or groups, or LTL segments or groups. In any case, the memory pool of the new I/O job does not have a block of memory large enough to allow the Loader to load these records.

E$NOSTART The target file does not specify the entry point for the program being loaded.

E$NOT$CONFIGURED This system call is not part of the present configuration.

E$NOUSER The calling task's job does not have a default user, or the object cataloged under the logical name R?IOUSER is not a user object.

E$PARAM At least one of the following is true:

- The value of the except$mode field within the except$handler structure lies outside the range Ø through 3.

- The target file requested a stack smaller than 16 bytes.

E$PATHNAME$-SYNTAX The specified pathname contains one or more invalid characters.

E$REC$FORMAT At least one record in the target file contains a format error.

DELETING THE COMMAND CONNECTION

After you have finished invoking commands programmatically, you must delete the command connection. The C$DELETE$COMMAND$CONNECTION system call performs this operation. You do not need to delete the command connection after each command invocation, because the command connection is re-usable. However, you should delete the command connection after performing all C$SEND$COMMAND operations. This frees the memory used by the data structures of the command connection.

EXAMPLE

Figure 5-1 contains an example of a program that uses C$CREATE$COMMAND$CONNECTION, SEND$COMMAND, and DELETE$COMMAND$CONNECTION. It invokes the Human Interface COPY command programmatically.

```
/************************************************************************
*                                                                        *
*   This example demonstrates the use of the following Human Interface   *
*   advanced standard functions:                                         *
*                                                                        *
*   rq$C$create$command$connection                                       *
*   rq$C$send$command                                                    *
*   rq$C$delete$command$connection                                       *
*                                                                        *
*   This program uses the previous system calls to invoke the command    *
*   COPY :F1:OLD to :F1:NEW from within and then continue normal         *
*   processing.  The program is invoked with the command line:           *
*                                                                        *
*       PROG2                                                            *
*************************************************************************/

prog2:  DO;

$include (hexcep.lit)
$include (hcrccn.ext)
$include (hsndcmd.ext)
$include (hdlccn.ext)
$include (iexioj.ext)
$include (hgtincn.ext)
$include (hgtocn.ext)

DECLARE (ci$token, co$token, command$connection$token)  WORD,
        (excep, comexcep, exexcep)  WORD;
DECLARE output$prep  BYTE;
```

Figure 5-1. Command Connection Example

```
                    .
                    .
                    .

/* Invoke utility to copy file OLD to file NEW */

/* Get tokens for CI and CO */
ci$token = rq$C$get$input$connection(@(4,':CI:'), @excep);
IF excep <> E$OK        THEN
    CALL rq$exit$io$job (excep, Ø, exexcep);
co$token = rq$C$get$output$connection(@(4,':CO:'), output$prep, @excep);
IF excep <> E$OK        THEN
    CALL rq$exit$io$job (excep, Ø, exexcep);

/* Create command connection */
command$connection$tok = rq$C$create$command$connection (ci$token,
                                                          co$token, Ø,
                                                          @excep);

/* Send command to copy files */
CALL rq$C$send$command (command$connection$tok,
                        @(23,'COPY :F1:OLD TO :F1:NEW'),
                        @comexcep, @excep);
IF excep <> E$OK        THEN
    CALL rq$exit$io$job (excep, Ø, exexcep);

/* Delete command connection */
CALL rq$C$delete$command$connection (command$connection$tok, @excep);
IF excep <> E$OK        THEN
    CALL rq$exit$io$job (excep, Ø, exexcep);

                    .
                    .         Rest of program
                    .

/* Finish I/O processing */
CALL rq$exit$io$job (excep, Ø, @exexcep);

END prog2;
```

Figure 5-1. Command Connection Example (continued)

***

Table B-3. Conditions And Their Codes (continued)

Category/ Mnemonic	Meaning	Numeric Code	
		Hex	Decimal
	Human Interface Environmental Conditions (continued)		
E$CONTINUED	The parse buffer contains a continuation character.	83H	131
E$INVALID$-NUMERIC	A numeric value contains invalid characters.	84H	132
E$LIST	A value in the value list is missing.	85H	133
E$WILDCARD	A wild-card character appears in an invalid context, such as in an intermediate component of a pathname.	86H	134
E$PREPOSITION	The command line contains an invalid preposition.	87H	135
E$PATH	The command line contains an invalid pathname.	88H	136
E$CONTROL$C	The user typed a CONTROL-C to abort the command.	89H	137
E$CONTROL	The command line contains an invalid control.	8AH	138
E$UNMATCHED$-LISTS	The number of files in the input and output pathname lists is not the same.	8BH	139
E$DATE	The operator entered an invalid date.	8CH	14Ø
E$NO$PARAM-ETERS	A command expected parameters, but the operator didn't supply any.	8DH	141
E$VERSION	The Human Interface is not compatible with the version of the command the operator invoked.	8EH	142
E$GET$PATH$-ORDER	A command called C$GET$OUTPUT$PATHNAME before calling C$GET$INPUT$PATHNAME	8FH	143
	UDI Environmental Conditions		
E$UNKNOWN$EXIT	The program exited normally.	ØCØH	192

Table B-3. Conditions And Their Codes (continued)

Category/ Mnemonic	Meaning	Numeric Code	
		Hex	Decimal
	UDI Environmental Conditions (continued)		
E$WARNING$EXIT	The program issued warning messages.	ØC1H	193
E$ERROR$EXIT	The program detected errors.	ØC2H	194
E$FATAL$EXIT	A fatal error occurred in the program.	ØC3H	195
E$ABORT$EXIT	The Operating System aborted the program.	ØC4H	196
E$UDI$INTERNAL	A UDI internal error occurred.	ØC5H	197
	Nucleus Programmer Errors		
* E$ZERO$-DIVIDE	A task attempted a divide in which the quotient was larger than 16 bits.	8ØØØH	32768
* E$OVERFLOW	An overflow interrupt occurred.	8ØØ1H	32769
E$TYPE	A token parameter referred to an existing object that is not of the required type.	8ØØ2H	3277Ø
E$PARAM	A parameter that is neither a token nor an offset has an invalid value.	8ØØ4H	32772
E$BAD$CALL	An OS extension received an invalid function code.	8ØØ5H	32773
* E$ARRAY$-BOUNDS	Hardware or software has detected an array overflow.	8ØØ6H	32774
* E$NDP$ERROR	A Numeric Processor Extension (NPX) error has occurred. OS extensions can return the status of the NPX to the exception handler.	8ØØ7H	32775
* E$ILLEGAL$-OPCODE	The iAPX 186 or 286 processor tried to execute an invalid instruction	8ØØ8H	32776

* For iAPX 286-based systems, a CPU trap caused this exceptional condition.

The key to using iRMX 86 files is the connection. A program wanting to use a file first obtains (a token for) a connection to the file and then uses the connection to perform operations on the file. Other programs can simultaneously have their own connections to the same file. Each program having a connection to a file uses its connection as if it has exclusive access to the file.

A program obtains a connection by calling DQ$ATTACH (if the file already exists) or DQ$CREATE (to create a new file). When the program no longer needs the connection, it can call DQ$DETACH to delete the connection. To delete both the connection and the file, the program calls DQ$DELETE.

Once a program has a connection, it can call DQ$OPEN to prepare the connection for input/output operations. The program performs input or output operations by calling DQ$READ and DQ$WRITE. It can move the file pointer associated with the connection by calling DQ$SEEK. When the program has finished doing input and output to the file, it can close the connection by calling DQ$CLOSE. Note that the program opens and closes the connection, not the file. Unless the program deletes the connection, it can continue to open and close the connection as necessary.

If a program calls DQ$DELETE to delete a file, the file cannot be deleted while other connections to the file exist. In that case, the file is marked for deletion and is not actually deleted until the last of the connections is deleted. During the time that a file is marked for deletion, no new connections to it may be created.

CONDITION CODES AND EXCEPTION HANDLING CALLS

Every UDI call except DQ$EXIT returns a numeric condition code specifying the result of the call. Each condition code has a unique mnemonic name by which it is known. For example, the code Ø, indicating that there were no errors or unusual conditions, has the name E$OK. Any other condition means there was a problem, so these conditions are called exceptions.

Exception conditions are classified as:

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

- Programmer Errors. These are typically caused by mistakes in programming (for example, "bad parameter"), but "divide-by-zero", "overflow", "range check", and errors detected by the 8Ø87 8Ø287 Numeric Processor Extension (hereafter referred to generically as the NPX) are also classified as programmer errors.

The iRMX 86 NUCLEUS REFERENCE MANUAL contains a list of condition codes that the iRMX 86 Operating System can return, with the mnemonic and meaning of each code.

If the default value (NEVER) for the EM parameter in the Nucleus ICU screen is in effect when a system call generates an exception condition, the system simply returns the error code through the appropriate system call parameter. If you have specified YES as the value of the EM parameter in the Nucleus ICU screen, the default system exception handler (DEF.EXCEPTIONHANDLER) displays the appropriate error message at the console and terminates the program. However, your program can establish its own exception handler by calling DQ$TRAP$EXCEPTION. The exception handler can interpret condition codes that are returned by calling DQ$DECODE$EXCEPTION. The rest of this section provides some facts that you need in order to write your own exception handler.

After an exception condition occurs and before your exception handler gets control, the iRMX 86 Operating System does the following:

1. Pushes the condition code onto the stack of the program that made the system call having the exception condition.

2. Pushes the number of the parameter that caused the exception onto the stack (1 for the first parameter, 2 for the second, etc.).

3. Pushes a word onto the stack (reserved for future use).

4. Pushes a word for the NPX onto the stack.

5. Initiates a long call to the exception handler.

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

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

MAKING UDI CALLS FROM PL/M-86 AND ASM86 PROGRAMS

This section describes how to make UDI calls from a program, using the DQ$ALLOCATE system call as an example. You can easily generalize from this example to see how to make the other UDI calls. There are two examples: one for a call from a PL/M-86 program and one for a call from an ASM86 program.

The way this chapter shows the DQ$ALLOCATE system call syntax is the following:

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

There are three parameters: size (which has the WORD data type), except$ptr (which has the POINTER data type), and base$addr (which has WORD data type or the SELECTOR data type, depending on the version of PL/M-86).

Each of the examples that follow request 128 bytes of memory and point to a WORD named "ERR" where the condition code is to be returned.

EXAMPLE PL/M-86 CALLING SEQUENCE

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

EXAMPLE ASM86 CALLING SEQUENCE

```
                  MOV     AX,128
                  PUSH    AX       ; first parameter
                  LEA     AX,ERR
                  PUSH    DS       ; second parameter
                  PUSH    AX       ;
                  CALL    DQALLOCATE
                  MOV     ARRAYBASE,AX ; returned value
```

This example is applicable to programs assembled according to the COMPACT, MEDIUM, and LARGE models of segmentation. For the SMALL model, omit pushing the DS segment register.

DESCRIPTIONS OF SYSTEM CALLS

This section contains descriptions of the UDI system calls, which are arranged alphabetically. Every system call description contains the following information in this order:

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

DQ$ALLOCATE

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

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

INPUT PARAMETER

size — A WORD which,

> if not zero, contains the size, in bytes, of the requested segment. If the size parameter is not a multiple of 16, it will be rounded up to the nearest multiple of 16 before the allocation request is processed.
>
> if zero, indicates that the size of the request is 65536 (64K) bytes.

OUTPUT PARAMETERS

base$addr — A SELECTOR, into which the Operating System places the base address of the memory segment. If the request fails because the memory requested is not available, this value will be ØFFFFH, and the system will return an E$MEM exception code.

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

DESCRIPTION

The DQ$ALLOCATE system call is used to request additional memory from the free space pool of the program. Tasks may use the additional memory for any desired purpose.

DQ$TRUNCATE

DQ$TRUNCATE moves the end-of-file to the current position of a named file connection's file pointer, thereby freeing the portion of the file lying beyond the file pointer.

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

INPUT PARAMETER

connection — A TOKEN for a connection to the named data file that is to be truncated. The file pointer of this connection marks the place where truncation is to occur. The byte indicated by the pointer is the first byte to be dropped from the file.

OUTPUT PARAMETER

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

DESCRIPTION

This system call truncates a file at the current setting of the file pointer and releases all file space beyond the pointer for reallocation to other files. If the pointer is at or beyond the end of file, no truncation is performed. Unless the file pointer is already at the proper location, your program should use the DQ$SEEK system call to position the pointer before calling DQ$TRUNCATE.

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

DQ$WRITE

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

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

INPUT PARAMETERS

connection — A TOKEN for the connection to the file into which the information is to be written.

buff$ptr — A POINTER to a buffer containing the data to be written to the specified file.

count — A WORD containing the number of bytes to be written from the buffer to the file.

OUTPUT PARAMETER

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

DESCRIPTION

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

Connection Requirements

If the connection is not open for writing or updating, DQ$WRITE returns an exception code.

***

CHAPTER 6
SIMPLIFYING CONFIGURATION DURING DEVELOPMENT

For your convenience, the configuration information found in this chapter has been added to the iRMX 86 CONFIGURATION GUIDE. For any information that you might need concerning the following topics, refer to the iRMX 86 CONFIGURATION GUIDE.

- Data segments
- Configuration
- Freezing locations of entry points
- The Interactive Configuration Utility (ICU)
- The LOC86 command
- Freezing the Base of the Data Segment

***

CHAPTER 2
USING A TERMINAL WITH THE iRMX™ 86 OPERATING SYSTEM

When you are using a terminal with the iRMX 86 Operating System, you must limit the maximum priority of your tasks or they could interfere with the proper functioning of your terminal. High priority processor-bound tasks can cause the Terminal Handler to drop input characters.

While using a terminal that is under control of the Terminal Handler, an operator either reads an output message from the terminal's display or enters characters by striking keys on the terminal's keyboard. Normal input characters are those destined for input messages that are sent to tasks. Special input characters direct the Terminal Handler to take special actions. The special characters are RUBOUT, Carriage Return, Line Feed, ESCape, control-C, control-O, control-Q, control-R, control-S, control-X, and control-Z. The output-only version of the Terminal Handler does not support any of the special characters. In the remainder of this chapter, the handling of these two types is discussed, and the significance of each of the special characters is explained.

NOTE

This chapter contains several references to mailboxes and request messages used by tasks to communicate with the terminal. If you are puzzled by such a reference, look in Chapter 3 for an explanation.

HOW NORMAL CHARACTERS ARE HANDLED

The destination of a normal character, when entered, depends on whether there is an input request message at the Terminal Handler's input request mailbox. If there is an input request message, the character is echoed to the terminal's display and goes into the input request message. If there is not an input request message, the character is deleted.

HOW SPECIAL CHARACTERS ARE HANDLED

Table 2-1 lists the special characters and summarizes the effects of each of them. The following text comprises complete descriptions of the effects of the special characters. In these descriptions, there are several references to "the current line." The current line consists of the data, with editing, that has been entered since the last end-of-line character.

Table 2-1. Special Character Summary

Special Character	Effect
RUBOUT	Deletes previously entered character.
Carriage Return	Signals end of line.
Line Feed	Signals end of line.
ESCape	Signals end of line.
control-C	Calls an application program.
control-O	Kills or restarts output.
control-Q	Resumes suspended output.
control-R	Displays current line with editing.
control-S	Suspends output.
control-X	Deletes the current line.
control-Z	Sends empty message.

The following descriptions concern the special characters needed when entering data at the terminal. Most of these characters are for line-editing. Each description is divided into two parts: internal effects and external effects. The difference is that internal effects are those that are not directly visible, whereas external effects are immediately shown on the terminal's display.

RUBBING OUT A PREVIOUSLY-TYPED CHARACTER (RUBOUT)

Internal Effects: Causes the most recently entered but not yet deleted character to be deleted from the current line. If the current line is empty, there is no internal effect.

CHAPTER 3
PROGRAMMING CONSIDERATIONS

The iRMX 86 Terminal Handler supports terminal input and output by providing mailbox interfaces. Figure 3-1 shows the use of these mailboxes. In the figure, an arrow pointing from a task to a mailbox represents an RQ$SEND$MESSAGE system call. An arrow pointing from a mailbox to a task indicates an RQ$RECEIVE$MESSAGE system call.

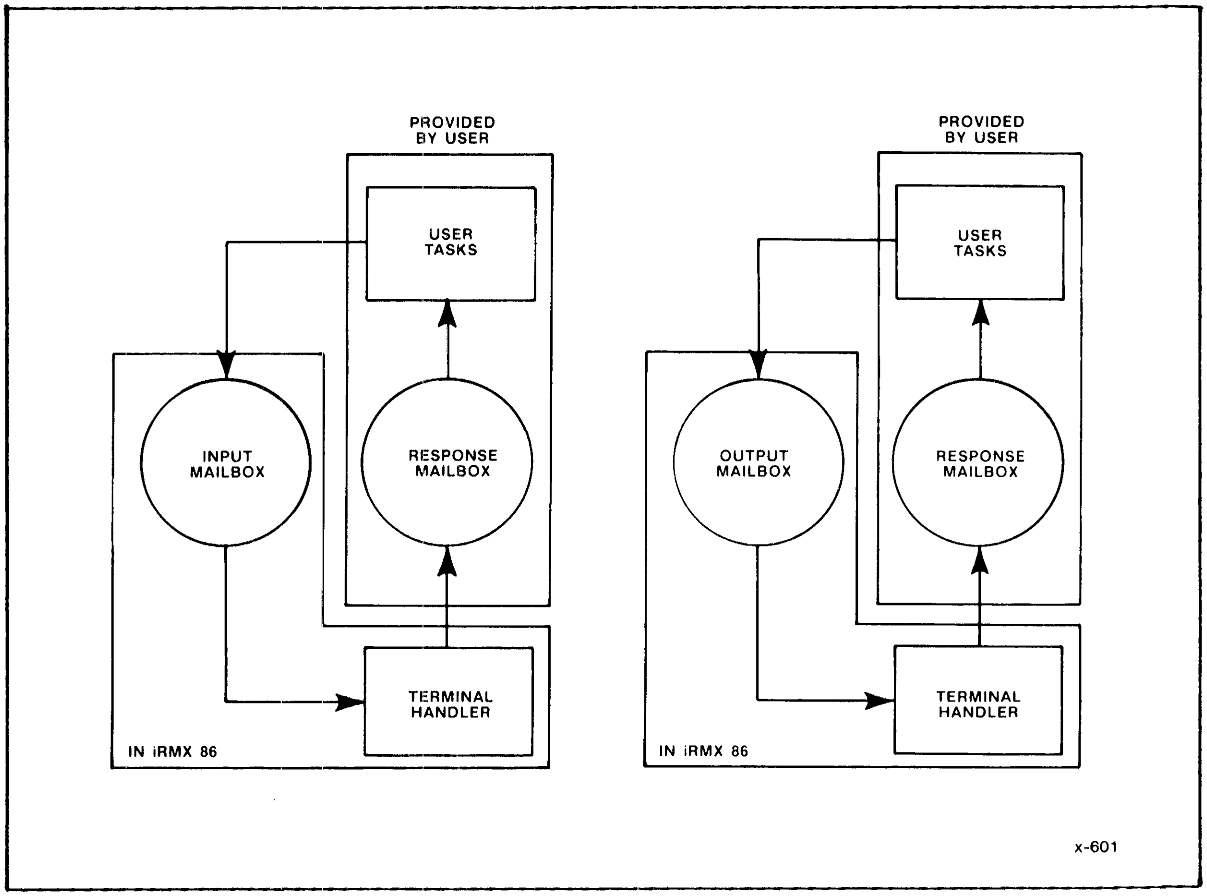


Figure 3-1. Input and Output Mailbox Interfaces

The protocol that tasks observe is much the same for input and output. In each case, the task initiates I/O by sending a request message to a mailbox. An input request mailbox (default name RQTHNORMIN) and an output request mailbox (default name RQTHNORMOUT) are provided. These mailboxes are cataloged in the root job directory. In the case of multiple terminals, one input and one output mailbox will be cataloged for each Terminal Handler. (See Chapter 4 for more information about multiple versions of the Terminal Handler.) Figure 3-2 illustrates the protocol for finding the root job token and for obtaining the input and output mailbox tokens.

```
/*****************************************************************
*  This example illustrates the protocol for finding the root job token *
*  and for obtaining the input and output mailbox tokens.           *
*****************************************************************

DECLARE rtjb$token                      WORD;
DECLARE root$job                        LITERALLY '3';
DECLARE status                          WORD;

DECLARE input$mbx$token                 WORD;

DECLARE wait$forever                    LITERALLY 'ØFFFFH';


/*By setting the input parameter to three, the GET$TASK$TOKEN primitive
  will return the root job's TOKEN.*/

rtjb$token = RQ$GET$TASK$TOKENS         (root$job,
                                        @status);

/*The following LOOKUP$OBJECT primitives use the default mailbox names.*/

input$mbx$token = RQ$LOOKUP$OBJECT      (rtjb$token,
                                        @(1Ø, 'RQTHNORMIN'),
                                        wait$forever,
                                        @status);

output$mbx$token = RQ$LOOKUP$OBJECT     (rtjb$token,
                                        @(11, 'RQTHNORMOUT'),
                                        wait$forever,
                                        @status);
```

Figure 3-2. Protocol for Obtaining Root Job and Mailbox Tokens

Refer to the iRMX 86 NUCLEUS REFERENCE MANUAL for more information concerning the individual primitives used in the previous example. When a task sends a message to the Terminal Handler mailbox, the Terminal Handler processes the request and then sends a response message back to the requesting task. The task waits at a response mailbox for the message. Thus, whether a task does input or output, it first sends and then receives. The full details of the input and output protocols are described later in this chapter. Output is discussed first because it is somewhat easier to understand.

For both input and output, a task sends a message segment to the Terminal Handler. The format of a request message is depicted in Figure 3-3. The numbers in that figure are offsets, in bytes, from the beginning of the segment. The field names have different meanings for input and for output. For both input and output, the first four fields are WORD values. The MESSAGE CONTENT field can be up to 132 bytes in length for input and up to 65527 bytes in length for output.

iRMX™ 86 Release 6.Ø Change Package: Update 3

Change Pages for:

iRMX™ 86 Installation and Configuration Guide (146197-ØØ1)

Table 6-7. Controller Board Switch Settings (continued)

Intel Board	Switch Setting	Description/Function
iSBC 220 (DIP Switches)	S1, 1-7 OFF 8 ON	Selects port address 100H.
	S2, 1-2 ON	Selects a 16-bit bus and 16-bit address decoding.
	3-10 OFF	Selects port address 100h.
iSBC 220 (Wire Wraps)	E16 - E15	Selects port address 100H
	E18 - E17 E20 - E19	Selects a 16-bit bus and 16-bit address decoding
iSBX 251		Not applicable.
iSBC 254		Not applicable.
iSBC 254S		Not applicable.
iSBX 270		Not applicable.
iSBX 351		Not applicable.
iSBC 534		Not applicable.
iSBC 544		If your board does not have a switch SW1, then refer to Table 6-3.
	SW1, 1-4 ON	Selects Dual-Port RAM address. Also refer to Table 6-4.
	SW1, 5 ON	Selects Dual-Port RAM size of 16K.
	SW1, 6 OFF	
	SW1, 7 ON	Selects 2732A EPROMS.
	SW1, 8 OFF	Configures board for slave mode.

DIP HEADER CONFIGURATIONS FOR THE RS232C PROTOCOL

Table 6-8 lists the DIP-header configurations you need to supply to implement the RS232C serial protocol. This configuration process involves either soldering wires on a solder style header or inserting wires into a pin-and-socket style header.

Table 6-8. DIP Header Configurations for the RS232C Protocol

Intel Board	DIP Header Jumpers	Description/Function
iSBX 351	3-13 4-14 7-8 5-6 11-12 9-1Ø	Board RxD to Terminal TxD. Board TxD to Terminal RxD. Board DSR to Board DTR. Board RTS to Board CTS. Terminal RTS to Terminal CTS. Terminal DSR to Terminal DTR.
iSBC 534	4-5 6-7 8-1Ø 9-11 12-13 14-15	Board DSR to Board DTR. Board RTS to Board CTS. Board RxD to Terminal TxD. Board TxD to Terminal RxD. Terminal RTS to Terminal CTS. Terminal DSR to Terminal DTR.
iSBC 544	2-3* 4-5 6-12 7-13 14-15 16-17*	Board DSR to Board DTR. Board RTS to Board CTS. Board RxD to Terminal TxD. Board TxD to Terminal RxD. Terminal RTS to Terminal CTS. Terminal DSR to Terminal DTR

Notes: Signal Names:

TxD:	Transmit Data	RxD:	Receive Data
DTR:	Data Terminal Ready	DSR:	Data Set Ready
RTS:	Request To Send	CTS:	Clear To Send

* If your terminal does not produce DSR but receives DTR, replace with the following: 2-16; 3-17

MISCELLANEOUS JUMPERS

Table 6-9 lists jumpering information not covered in the previous sections. The list of jumpers change different functional areas. Perform the changes to use default values established by Intel.

Table 6-9. Miscellaneous Jumpers

Intel Board	Remove Jumper	Add Jumper	Description/Function
iSBC 2Ø4	E75-E76 E77-E78		Use if iSBC 2Ø4 has two 8271 devices installed.
iSBC 2Ø6			None.
iSBC 2Ø8			None.
iSBC 215	W4, 1-2		Remove only if installing an iSBX 218A in iSBX socket 1 (J4).
iSBC 215G	W4, 1-2		Remove only if installing an iSBX 218A in iSBX socket 1 (J4).
		W24, 1-2	Use if installing an iSBX 218(A) in iSBX socket 1 (J4).
		W2Ø, 1-2	Connects -12 volts from the MULTIBUS to the iSBC 215G.
iSBX 218	W1, A-B	W1, A-C	Disables iSBX 218 DMA lines.
iSBX 218A			This board may require some special jumper changes depending on the requirements of your application. Consult the iSBX 218A Hardware Reference Manual for special considerations.
iSBC 22Ø			None.
iSBX 251			None.
iSBC 254			None.
iSBC 254S			None.

Table 6-9. Miscellaneous Jumpers (continued)

Intel Board	Remove Jumper	Add Jumper	Description/Function
iSBX 27Ø	E11-E12 E16-E17 E21-E22 E23-E24		Sets up your terminal screen output.
iSBX 351			None.
iSBC 534			None.
iSBC 544			None.

***

CHAPTER 10
DEVICE DRIVER PARAMETERS

This chapter discusses how to respond to the prompts that appear on the Intel Device Driver screens. If you are using this chapter to understand a particular parameter line, search Table 1Ø-1 for the device driver that interests you and then turn to the page indicated to the right of the device driver.

Table 1Ø-1. Intel-Supplied Device Drivers

Device Driver	Page Number
iSBC 2Ø4	1Ø-Ø2
iSBC 2Ø6	1Ø-12
iSBC 2Ø8	1Ø-21
iSBC 215	1Ø-34
iSBX 218	1Ø-5Ø
iSBC 22Ø	1Ø-62
iSBC 254	1Ø-76
iSBX 27Ø	1Ø-86
iSBC 534	1Ø-95
iSBC 544	1Ø-1Ø6
8251A Terminal Driver	1Ø-118
Line Printer	1Ø-13Ø
USART Terminal Handler Driver	1Ø-133
8274 Terminal Driver	1Ø-135
Line Printer for iSBC 286/1Ø	1Ø-152
iSBC 188/48	1Ø-152.1
iSBX 251	1Ø-156
SCSI Driver for iSBC 186/Ø3	1Ø-164
iSBX 218A	1Ø-179
RAM Driver	1Ø-191
iSBC 216	1Ø-2ØØ
8253Ø Terminal Driver	1Ø-2Ø1

If you are adding a user-supplied device driver, refer to page 1Ø-214.

iSBC↓ 2Ø4 DRIVER PARAMETERS

The iSBC 2Ø4 flexible disk driver:

- Supports 8-inch, single-sided, single-density diskettes.

- Supports the READ, WRITE, SEEK, SPECIAL, ATTACH$DEVICE, and DETACH$DEVICE functions.

- Accepts functions OPEN and CLOSE but performs no operations for them.

Track formatting and volume change notification are supported via the SPECIAL function. Refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for further information about these special functions.

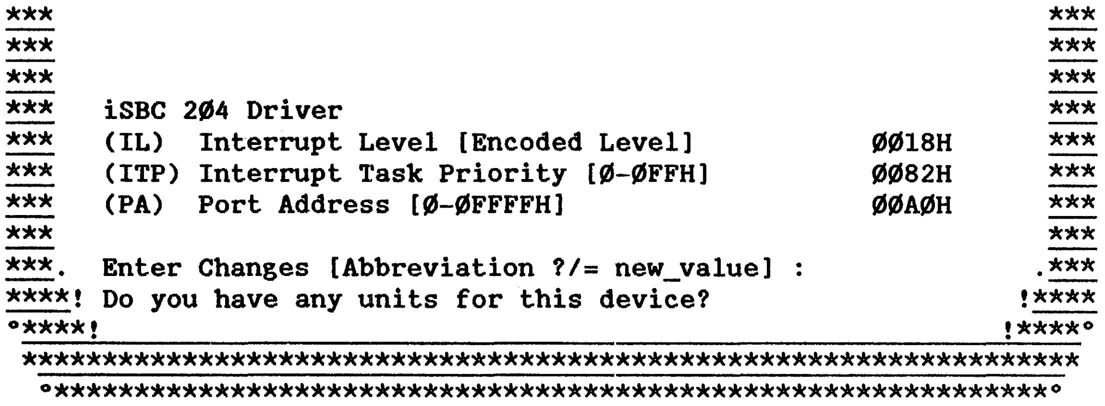
The iSBC 2Ø4 driver supports up to four units per controller, two for each 8271 flexible disk controller component. The typical controller has one 8271 component. This component supports two single-sided units.

There are three screens that define the interface between the iSBC 2Ø4 random access device driver and the I/O system. These screens relate to the three device configuration tables: the device information table, the unit information table, and the device unit information block (DUIB). Refer to Appendix D for further information about these tables.

The values shown on the screens in this section are the same as values you would see if you choose option "Ø" from the Intel-supplied device driver screen.

iSBC↓ 2Ø4 DRIVER SCREEN

The ICU uses the information from the following screen to create a device information table for the iSBC 2Ø4 driver. If your system includes more than one iSBC 2Ø4 controller, you must specify a unique interrupt level and port address for each controller.



```
***                                                            ***
***                                                            ***
***                                                            ***
***   iSBC 2Ø4 Driver                                          ***
***   (IL)  Interrupt Level [Encoded Level]          ØØ18H     ***
***   (ITP) Interrupt Task Priority [Ø-ØFFH]         ØØ82H     ***
***   (PA)  Port Address [Ø-ØFFFFH]                  ØØAØH     ***
***                                                            ***
***.  Enter Changes [Abbreviation ?/= new_value] :            .***
****! Do you have any units for this device?                 !****
°****!                                                      !****°
 *****************************************************************
  °*************************************************************°
```

pool, so by setting this parameter to 0FFH you allow the calling job to select the number of buffers based on its own memory pool size. It is recommended that you use the default value.

```
************************************************************************
*        Do you have any more DUIBs for this device?                   *
************************************************************************
```

Respond "Yes" to this prompt "Do you have any more device-unit information blocks for this device?" if you plan to use the iSBC 208 controller with two devices that have different characteristics.

While developing your initial systems, you can create as many device-unit information blocks as you want. The number of DUIBs can exceed the number of devices on your system. The particular DUIB associated with the device depends on the physical name you use when attaching it. Once you know that you will never need a particular DUIB, save memory by deleting it from your description file before you generate your configuration files (refer to Chapter 17 for additional information on generating configuration files).

iSBC↓ 215 DRIVER PARAMETERS

The iSBC 215 Winchester disk driver:

- Supports the READ, WRITE, SEEK, SPECIAL, ATTACH$DEVICE, and DETACH$DEVICE functions.

- Accepts the OPEN and CLOSE functions but performs no operations for it.

Track formatting and volume change notification are supported via the SPECIAL function. Refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for further information about these special functions.

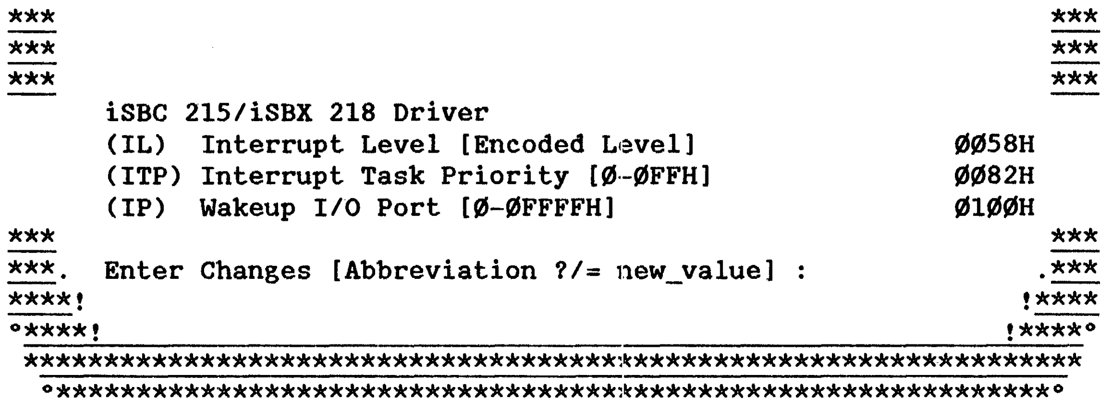
There are three screens that define the interface between the iSBC 215 random access device driver and the I/O system. These screens relate to the three device configuration tables: the device information table, the unit information table, and the device unit information block (DUIB). Refer to Appendix D for further information about these tables.

The three screens that are described in this section are labeled "iSBC 215/218". This means that the screen supports both the iSBC 215 and the iSBX 218 controllers. This section describes only the iSBC 215-related parameter lines. Refer to the section of this chapter labeled "iSBX™ 218 DRIVER PROMPTS" for information on how to respond to iSBX 218-related parameter lines.

The values shown on the screens in this section are the same as values you would see if you use the rmx86.def file when you invoke the ICU.

iSBC↓ 215 DRIVER SCREEN

The ICU uses the information from the following screen to create a device information table for the iSBC 215 driver. If your system includes more than one iSBC 215 controller, you must specify a unique interrupt level and wakeup I/O port address for each controller.



```
iSBC 215/iSBX 218 Driver
(IL)  Interrupt Level [Encoded Level]                 ØØ58H
(ITP) Interrupt Task Priority [Ø-ØFFH]                ØØ82H
(IP)  Wakeup I/O Port [Ø-ØFFFFH]                      Ø1ØØH

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

==

iRMX™ 86 OPERATING SYSTEM RELEASE 6 CHANGE PACKAGE: UPDATE 2

==

Purpose

The change pages in this package correct technical errors identified in the current version of the iRMX™ 86 Release 6 documentation.

Scope

The following manuals are affected by this change package:

Introduction and Operator's Reference Manual (146194-ØØ1)
Programmer's Reference Manual, Part I (146195-ØØ1)
Programmer's Reference Manual, Part II (146196-ØØ1)
iRMX™ 86 Installation and Configuration Guide (146197-ØØ1)

Installation Instructions

Change pages in the Update Package are accumulated from quarter to quarter. The change pages for each successive update are separated in this package by a blue cover page (similar to the sheet you are now reading). Within each update section, yellow, pink, green, and orange cover sheets segregate the change pages according to volume.

The change pages in this package are installed by removing a page from your documentation and replacing it with the corresponding page from the change package.

If this is the first iRMX™ 86 Release 6.Ø Update to be installed in your documentation:

1. Install the change pages in this section before installing the change pages for Update 3.

If you have installed previous iRMX™ 86 Release 6.Ø Updates in your documentation:

1. Discard this section.

iRMX™ 86 Release 6.Ø Change Package: Update 2

Change Pages for:

iRMX™ 86 Introduction and Operator's Reference Manual (146194-ØØ1)

Another advantage of hierarchical file structure is that duplicate file names are permitted unless the files reside in the same directory. Notice in Figure 2-2 that the file tree contains two directories named BILL. (These directories are on the extreme left and extreme right of the figure.) However, the Operating System recognizes them as unique files because each resides in a different directory.

Each file tree resides on a secondary storage volume -- the storage medium that contains the data. Examples of volumes include flexible diskettes, hard disks, and bubble memories. Before you can place named files on a volume, you must format the volume to accept named files. The formatting process writes a number of data structures on the volume to aid the Operating System in creating and maintaining files. You can use the FORMAT command (described in Chapter 3) to format your volumes.

The uppermost point of each file tree is a directory called the root directory. When formatted for named files, each secondary storage volume has one and only one root directory. For these reasons:

- There can be only one file tree per secondary storage volume.
- A file tree cannot extend to more than one volume.

PATHNAMES

This section describes how to specify a particular file in a named-file tree. For simplification, it assumes that all files reside in the same file tree, and thus in the same volume. To identify the volume as well as the file, you must include a logical name for the device as the first portion of the file specification. Refer to the "Logical Names" section, later in this chapter, for more information about logical names.

In a file tree, each file (data or directory) has a unique shortest path connecting it to the root directory. For example, in Figure 2-2, the shortest path from the root directory to file BATCH-2 goes through directory DEPT1, through directory TOM, through directory TEST-DATA, and finally stops at data file BATCH-2. When you want to perform an operation on a file (for example, using the COPY command to copy one file to another), you must specify not only the file's name, but the path through the file tree to the file. This description is called the file's pathname. For file BATCH-2 in Figure 2-2, the pathname is:

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

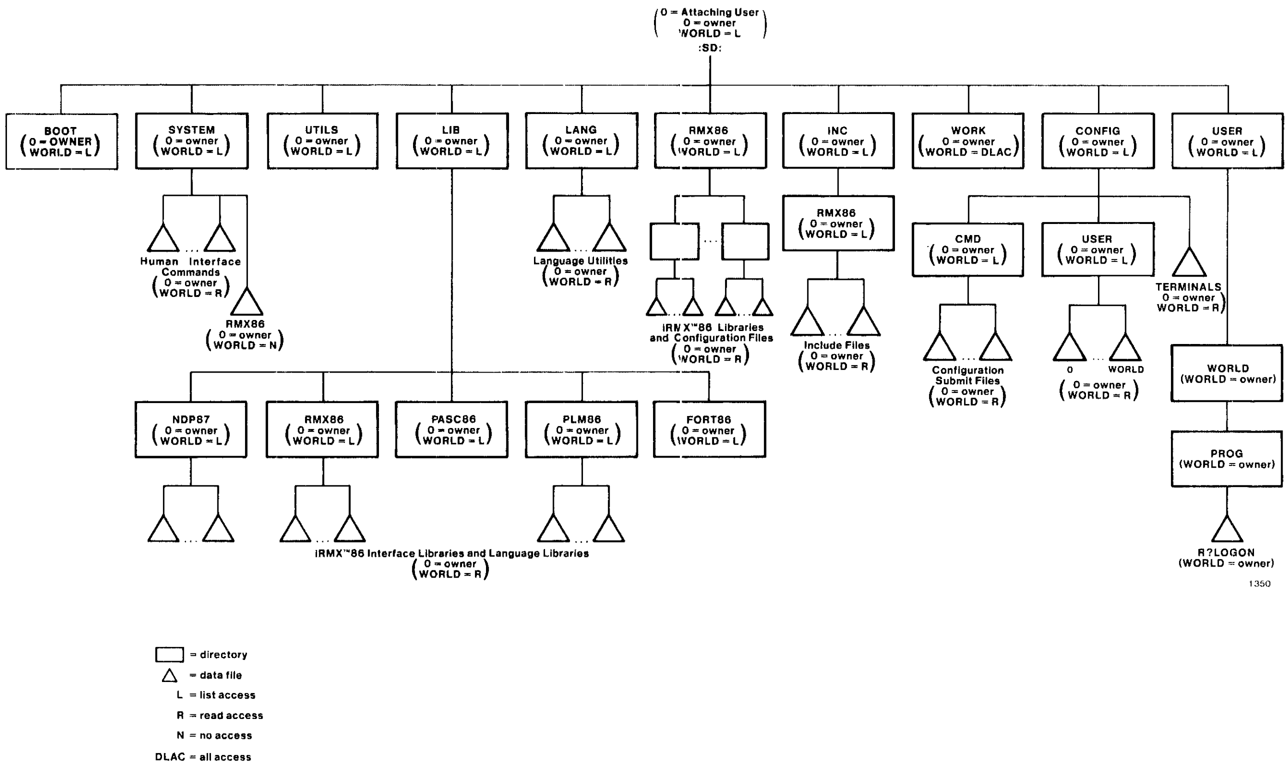


Figure 2-3. File Structure on an Intel Supplied Start-Up System

- A task deletes the connection to the file via a Basic I/O System or Extended I/O System call (refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL or the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL for more information about connections). In this instance, the logical name remains cataloged in the global directory, but the connection to which it refers does not exist.

- A user forcibly detaches the volume containing the file via the DETACHDEVICE command (described later in this chapter).

- A user removes the volume from the drive.

ERROR MESSAGES

- <logical name>, list of logical names not allowed

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

- <pathname>, list of pathnames not allowed

  You entered more than one pathname as input to ATTACHFILE.

- <logical name>, logical name not allowed

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

- <logical name>, not a file connection

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

- <pathname>, not allowed as default prefix

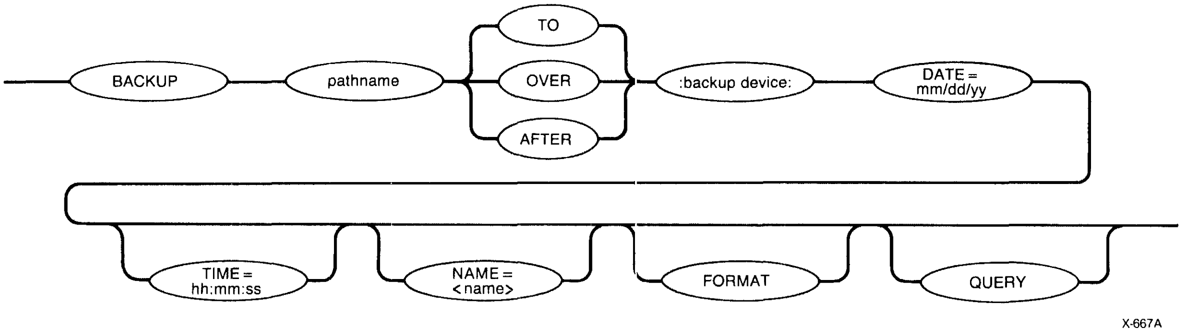
  You attempted to attach a physical or stream file as your default prefix (:$:). Only named files are valid.

- <logical name>, too many logical names

  Your global object directory is full. Therefore ATTACHFILE is unable to catalog the file's name in the object directory.

BACKUP

This command saves files on a named volume by copying them to a physical volume which serves as a backup storage device. This command provides a way of saving a large volume (a Winchester disk, for example) onto a number of smaller volumes such as diskettes or onto another mass storage device such as a tape drive. Later, you can use the RESTORE command (described later in this chapter) to retrieve these files and copy them to a named volume.



INPUT PARAMETERS

pathname — Pathname of a file on the source volume. BACKUP saves all the files starting from this point on the file tree. If you specify the logical name of the device only, BACKUP saves all files in the volume, beginning with the root directory. If you specify a file and not a directory, then only the specified file is saved.

DATE — BACKUP saves all files created or modified on or after the date and time specified with the DATE/TIME parameters. If the DATE parameter is omitted, the date defaults to the current system date. If both date and time parameters are omitted (DATE/TIME), then the date and time default to 1/1/78 and 00:00:00.

mm/dd/yy — Form used to specify the DATE.

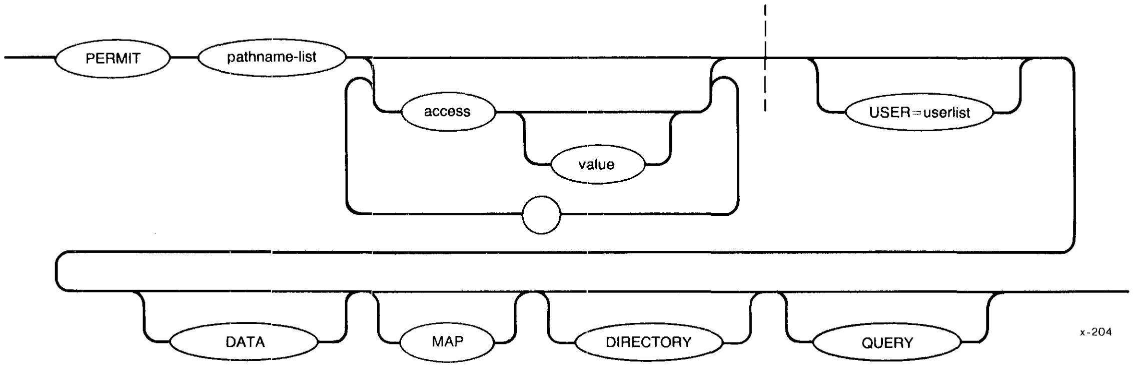
mm — Numerical designation for the month. (For example: 1 represents January, 2 represents February, etc.). Must be a digit.

dd — Numerical designation for the day of the month. Value must be in digits.

yy — Designation for the year. You enter this as a two digit number, as follows:

PERMIT

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



INPUT PARAMETERS

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

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

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

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

value

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

Value	Meaning
0	Rescind the access right
1	Grant the access right

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

user-list

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

WORLD Special user ID (0FFFFh) giving all users access to the file.

* Designator indicating that the access rights apply to all users currently in the file's access list.

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

DATA

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

DIRECTORY

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

MAP

Specifies that access information also applies to the map and volume label files in the pathname list. If you use the MAP parameter, you must specify the full pathname of any map files or volume label files in the pathname list. For, example PERMIT :f0:R?* DLAU MAP will change the access rights for all map files and volume label files on the volume (with the exception of R?SAVE which is unaffected by the MAP parameter). Notice that, in this instance, the Human Interface does not interpret the "?" as a wild card character.

QUERY

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

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

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

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

DESCRIPTION

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

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

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

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

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

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

```
A0DU0
A0,D,U0
A0D1U0
A0,D1,U0
```

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

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

```
D0,A1,R1,D1
```

is the same as the combination:

```
A1,R1,D1
```

In the first combination, the D1 overrides the D0.

You can use the N character to rescind all access to the file. If specified alone, it removes user IDs from the accessor list. However, the N character can also be useful when changing access rights, if you don't remember the specified user's current access rights. In this case you can specify the N character first, to clear all the access rights, and follow it with other characters to grant the desired access. For example, if you want to grant list access only, you could specify "NL" instead of "D0A0C0L".

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

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

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

```
-L-C
```

indicates that the corresponding user has list and change access.

- output specification missing

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

- <pathname>, READ access required

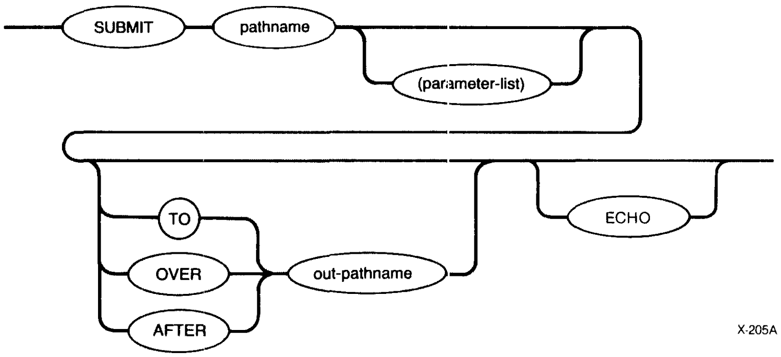
  You do not have read access to a file on the backup volume; therefore RESTORE cannot restore the file.

- <pathname>, too many input pathnames

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

SUBMIT

This command reads and executes a set of commands from a file in secondary storage instead of from the console keyboard.



INPUT PARAMETERS

pathname — Name of the file from which the commands will be read. This file may contain nested SUBMIT commands.

parameter-list — Actual parameters that are to replace the formal parameters in the SUBMIT file. You must surround this parameter list with parentheses. You can specify as many as 10 parameters, separated by commas, in the SUBMIT command. If you omit a parameter, you must reserve its position by entering a comma. If a parameter contains a comma, space, or parenthesis, you must enclose the parameter in single quotes. The sum of all characters in the parameter list must not exceed 512 characters.

OUTPUT PARAMETERS

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

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

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

OVER

Causes the output for each command in the SUBMIT file to be written over the specified existing file instead of the console screen.

AFTER

Causes the output from each command in the SUBMIT file to be written to the end of an existing file instead of the console screen.

out-pathname

Pathname of the file to receive the processed output from each command executed from the SUBMIT file. If no preposition or output file is specified, TO :CO: is the default.

ECHO

ECHO causes the a copy of the data read from the first level of a SUBMIT file to be sent to the CRT. This parameter lets you know which action specified within a SUBMIT file is currently executing. Nested SUBMIT commands do not have their contents sent to the console.

DESCRIPTION

To use the SUBMIT command you must first create a data file that defines the command sequence and formal parameters (if any). The Operating System first looks for the pathname ending in "CSD". If no such file is found, then the Operating System looks for the specified file in the pathname.

Any program that reads its commands from the console input (:CI:) can be executed from a SUBMIT file. If another SUBMIT command is itself used in a SUBMIT file, it causes another SUBMIT file to be invoked. You can nest SUBMIT files to any level of nesting until memory is exhausted (each level of SUBMIT requires approximately 10K of dynamic memory). When one nested SUBMIT file completes execution, it returns control to the next higher level of SUBMIT file.

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

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

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

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

```
END SUBMIT <pathname>
```

You may use SUPER sub-commands (such as CHANGEID) within a SUBMIT file. To do so, you must include a SUPER command in the SUBMIT file. The SUPER command must precede any of the sub-commands in the file. When the SUBMIT file encounters the SUPER command, you are prompted for a password. Execution of the remainder of the file does not resume until you respond. You can avoid interrupting execution of the SUBMIT file by invoking the file while you are in the SUPER mode. In this case, the SUBMIT file still requires an embedded SUPER command. However, you are not prompted to re-enter the password when the SUBMIT file executes.

ERROR MESSAGES

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

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

- <parameter>, incorrectly formed parameter

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

- <pathname>, output file same as input file

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

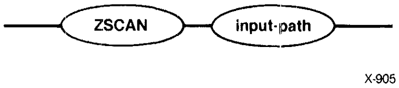
- <pathname>, too many input files

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

- <parameter>, too many parameters

ZSCAN

This command reads an object file or an object library and displays the Identification Number of all ZAPs that have been applied to that file.



INPUT PARAMETER

input-path	The pathname of the object file or object library to be scanned. The pathname cannot contain wildcard characters. The pathname must specify a file, not a directory.

DESCRIPTION

Fixes for problems discovered in the operation system software are distributed through the iRMX 86 Update Service. Intel refers to these fixes as "ZAPs". ZAPs are patched modules that replace the corresponding module in the operation system.

Each update diskette contains an accumulation of all ZAPs assured during the current release of the operating system. When you install the latest update, all ZAPs (from the current update and from previous updates) are automatically applied to your system.

The ZSCAN command allows you to check which ZAPs have been applied to an object file or an object library. Beginning with iRMX 86, Release 6.0, Update 2.0 all ZAPs (including all ZAPs from previous Release 6 updates) are marked by a unique identifier string. Installing Update 2 -- or any later update -- assures that ZAP identifier strings are affixed to all ZAPs currently applied to iRMX 86 Release 6.0. ZSCAN finds occurrences of there strings and returns information about the associated ZAPs.

When you invoke ZSCAN, you must specify an object file or an object library. You can not invoke the command to find all of the ZAPs applied within a specified directory. Furthermore, you cannot use wildcard characters in the pathname of the file to be scanned.

Output from ZSCAN is automatically directed to your terminal. To re-direct output to any other destination, place the ZSCAN command line in a SUBMIT file. Then invoke the SUBMIT file specifying the desired output destination.

By default, the iRMX 86 system object files are not accessible to user WORLD. Therefore, if you intend to use ZSCAN on a bootable system object file, you must grant user WORLD read access rights to that file (using the Human Interface PERMIT command) or invoke ZSCAN from the SUPER mode.

OUTPUT DISPLAY

Upon successful execution, the ZSCAN command displays one of the two following messages.

When ZSCAN encounters ZAPs:

```
< filename > , has the following ZAP(s) applied:
< zap id > , < class > : for iRMX 86 R6.0, < layer > < version >
                                 .
                                 .
                                 .
< zap id > , < class > : for iRMX 86 R6.0, < layer > < version >
```

where:

<filename> the name of the file being scanned.

<zap id> the identification code for the ZAP:

```
Z  BR  ⎡A⎤xx
       ⎣B⎦
```

where:

BR = a iRMX 86 Release 6.0 ZAP;

A = a Class A ZAP
B = a Class B ZAP

xx = a unique ID number from 0 to 99

<class> the class of the ZAP. Class A indicates a supported ZAP distributed through the iRMX 86 update service. Class B indicates an un-supported ZAP with limited distribution.

<layer> the layer of the operating system (e.g. Nucleus, BIOS, etc.) that the ZAP pertains to.

<version> the version of the operating system layer that the ZAP pertains to.

When ZSCAN encounters no ZAPs:

```
< filename > , No ZAPs applied
```

where:

< filename >   the name of the file bing scanned.

ERROR MESSAGES

- < filename > file does not exist.

  There is no file with the pathname specified in the command.

- < filename > is not an object module.

  The file specified in the command is not an object module and thus cannot be scanned for ZAPs.

<fnodenum>, fnode out of range

The fnode number that you specified was larger than the largest fnode number in the volume.

no badblocks file

Your system does not have a bad blocks file. This message could appear because you used a Release 4 or earlier version of the Human Interface command, FORMAT, when you formatted your disk.

DISK COMMAND

This command displays the attributes of the volume being verified. You can abort this command by typing a CONTROL-C (press the CONTROL key, and while holding it down, press the C key). The format of the DISK command is as follows:

x-225

OUTPUT

The output of the DISK command depends on whether the volume is formatted as a physical or named volume. For a physical volume, the DISK command displays the following information:

```
     Device name = <devname>
   Physical disk
     Device gran = <devgran>
      Block size = <devgran>
    No of blocks = <numblocks>
     Volume size = <size>
```

where:

<devname>   Name of the device containing the volume. This is the physical name of the device, as specified in the ATTACHDEVICE Human Interface command.

<devgran>   Granularity of the device, as defined in the device unit information block (DUIB) for the device. Refer to the iRMX 86 CONFIGURATION GUIDE for more information about DUIBs. For physical devices, this is also the volume block size.

<numblocks>   Number of volume blocks in the volume.

<size>   Size of the volume, in bytes.

For a named volume, the DISK command displays the following information:

```
                Device name = <devname>
    Named disk, Volume name = <volname>
                Device gran = <devgran>
                 Block size = <volgran>
               No of blocks = <numblocks>
          No of Free blocks = <numfreeblocks>
                Volume size = <size>
                 Interleave = <inleave>
             Extension Size = <xsize>
               No of fnodes = <numfnodes>
          No of Free fnodes = <numfreefnodes>
```

If the formatting program is unable to provide this information, it places an ASCII space in this field.

- The next two bytes contain a two-digit ASCII sequence number which is incremented by the formatting program each time the formatting program changes in a way that affects the volume format. The Release 4 FORMAT Human Interface command places the characters "00" in this field.

- The right-most three bytes of the field contain a three-digit ASCII number specifying the version of the Basic I/O System that was used in formatting the volume (for example, the characters "030" would indicate version 3.0). If the formatting program is unable to obtain this information, it places ASCII spaces in this field.

DEVICE$SPECIAL(8) Reserved for special device-specific information. When no device-specific information exists, this field must contain zeros. If the device is a Winchester disk with an iSBC 215 controller or if the device is a disk with an iSBC 220 controller, the iRMX 86 Operating System imposes a structure on this field and supplies the following information:

```
SPECIAL                 STRUCTURE(
CYLINDERS               WORD,
FIXED                   BYTE,
REMOVABLE               BYTE,
SECTORS                 BYTE,
SECTOR_SIZE             WORD,
ALTERNATES              BYTE);
```

where:

CYLINDERS Total number of cylinders on the drive.

FIXED Number of heads on the fixed disk or Winchester disk.

REMOVABLE Number of heads on the removable disk cartridge.

SECTORS Number of sectors in a track.

SECTOR_SIZE Sector size, in bytes.

ALTERNATES Number of alternate cylinders.

The remainder of the Volume Label (bytes 430 through 511) is reserved and must be set to zero.

INITIAL FILES

Any mechanism that formats iRMX 86 named volumes must place seven files on the volume during the format process. These seven files are the fnode file, the volume label file, the volume free space map file, the free fnodes map file, the bad blocks file, the root directory, and the space accounting file. The first of these files, the fnode file, contains information about all of the files on the volume. The general structure of the fnode file is discussed first. Then all of the files are discussed in terms of their fnode entries and their functions.

FNODE FILE

A data structure called a file descriptor node (or fnode) describes each file in a named file volume. All the fnodes for the entire volume are grouped together in a file called the fnode file. When the I/O System accesses a file on a named volume, it examines the iRMX 86 Volume Label (described in the previous section) to determine the location of the fnode file, and then examines the appropriate fnode to determine the actual location of the file.

When a volume is formatted, the fnode file contains seven allocated fnodes and any number of un-allocated fnodes. The original number of un-allocated fnodes depends on the FILES parameter of the FORMAT command. These allocated fnodes represent the fnode file, the volume label file, the volume free space map file, the free fnodes map file, the bad blocks file, the root directory, and the space accounting file. Later sections of this chapter describe these files. The size of the fnode file is determined by the number of fnodes that it contains. The number of fnodes in the fnode file also determines the number of files that can be created on the volume. The number of files is set when you format the storage medium.

The structure of an individual fnode in a named file volume is as follows:

```
    DECLARE
            FNODE          STRUCTURE(
                   FLAGS                WORD,
                   TYPE                 BYTE,
                   GRAN                 BYTE,
                   OWNER                WORD,
                   CR$TIME              DWORD,
                   ACCESS$TIME          DWORD,
                   MOD$TIME             DWORD,
                   TOTAL$SIZE           DWORD,
                   TOTAL$BLKS           DWORD,

                   POINTR(40)           BYTE,

                   THIS$SIZE            DWORD,
                   RESERVED$A           WORD,
                   RESERVED$B           WORD,
                   ID$COUNT             WORD,

                   ACC(9)               BYTE,
                   PARENT               WORD,
                   AUX(*)               BYTE);
```

INDEX

Underscored entries are primary references.

iRMX™ 86 Release 6.Ø Change Package: Update 2

Change Pages for:

iRMX™ 86 Programmer's Reference Manual, Part I (146195-ØØ1)

- A reserved (WORD) parameter.
- A (WORD) parameter containing the Numeric Processor Extension (NPX) status word. This parameter is valid only if the condition code is E$NDP$ERROR.

ASSIGNING AN EXCEPTION HANDLER

A task may use the SET$EXCEPTION$HANDLER system call to declare its own exception handler. Otherwise, the task inherits the exception handler of its job. A job can receive its own exception handler at the time of its creation. If it doesn't, the job inherits the system exception handler. Thus, the Nucleus can always find an exception handler for the running task.

A system exception handler is provided as part of the iRMX 86 Operating System. Depending on a configuration option, it either deletes or suspends any task on whose behalf it is invoked. The iRMX 86 CONFIGURATION GUIDE describes this configuration option.

Users wanting to write their own exception handlers should compile them under the PL/M-86 LARGE control, specifying the PUBLIC attribute.

Any task can have the Debugger as its exception handler; see the description in Chapter 12 of the SET$EXCEPTION$HANDLER system call for instructions on how to dynamically make such an assignment. Alternatively, the Debugger or any other routine can be made the system exception handler statically; see the iRMX 86 CONFIGURATION GUIDE for information on how to do this.

INVOKING AN EXCEPTION HANDLER

An exception handler normally receives control when an exceptional condition occurs. However, when a task encounters an exceptional condition, it need not always have control passed to its exception handler. The factor that determines whether control passes to the exception handler is the task's exception mode. This attribute has four possible values, each of which specifies the circumstances under which the exception handler is to get control in the event of an exceptional condition. These circumstances are:

- Programmer errors only.
- Environmental conditions only.
- All exceptional conditions.
- No exceptional conditions.

When the Nucleus detects that a task has caused an exceptional condition in making a system call, it compares the type of the condition with the calling task's exception mode. If a transfer of control is indicated, the Nucleus passes control to the exception handler on behalf of the task. The exception handler then deals with the problem, after which control returns to the task, unless the exception handler deleted the task. When the exception handler returns, the task can also detect that an error occurred, because the system call's except$ptr parameter points to a word containing the condition code. While the exception handler is executing, the errant task is still regarded by the Nucleus to be the running task.

When a task is created, its exception mode is set to its job's default exception mode. The task can change its exception handler and exception mode attributes by using the SET$EXCEPTION$HANDLER system call.

HANDLING EXCEPTIONS IN-LINE

If a task's exception mode attribute does not direct the Nucleus to transfer control to the task's exception handler, the responsibility for dealing with an error falls upon the task.

Each system call has as its last parameter a POINTER to a WORD. After a system call, the Nucleus returns the resulting condition code to this WORD. By checking this WORD after each system call, a task can ascertain whether the call was successful. (See Table 7-1 for condition codes.) If the call was not successful, the task can learn which exceptional condition it caused. This information can sometimes enable the task to recover. In other cases more information is needed.

If a system call returns an exception code to indicate an unsuccessful call, all other output parameters of that system call are undefined.

NOTE

> If an exceptional condition is caused by an invalid parameter, an exception handler, which is passed the parameter number of the first invalid parameter, should handle the condition.

HANDLING EXCEPTIONS IN iAPX 286-BASED SYSTEMS

The Operating System software catches and returns most of the exceptional conditions listed in Table 7-1. However, a few conditions (those noted with asterisks in the table) occur because the microprocessor catches (or traps) an invalid condition.

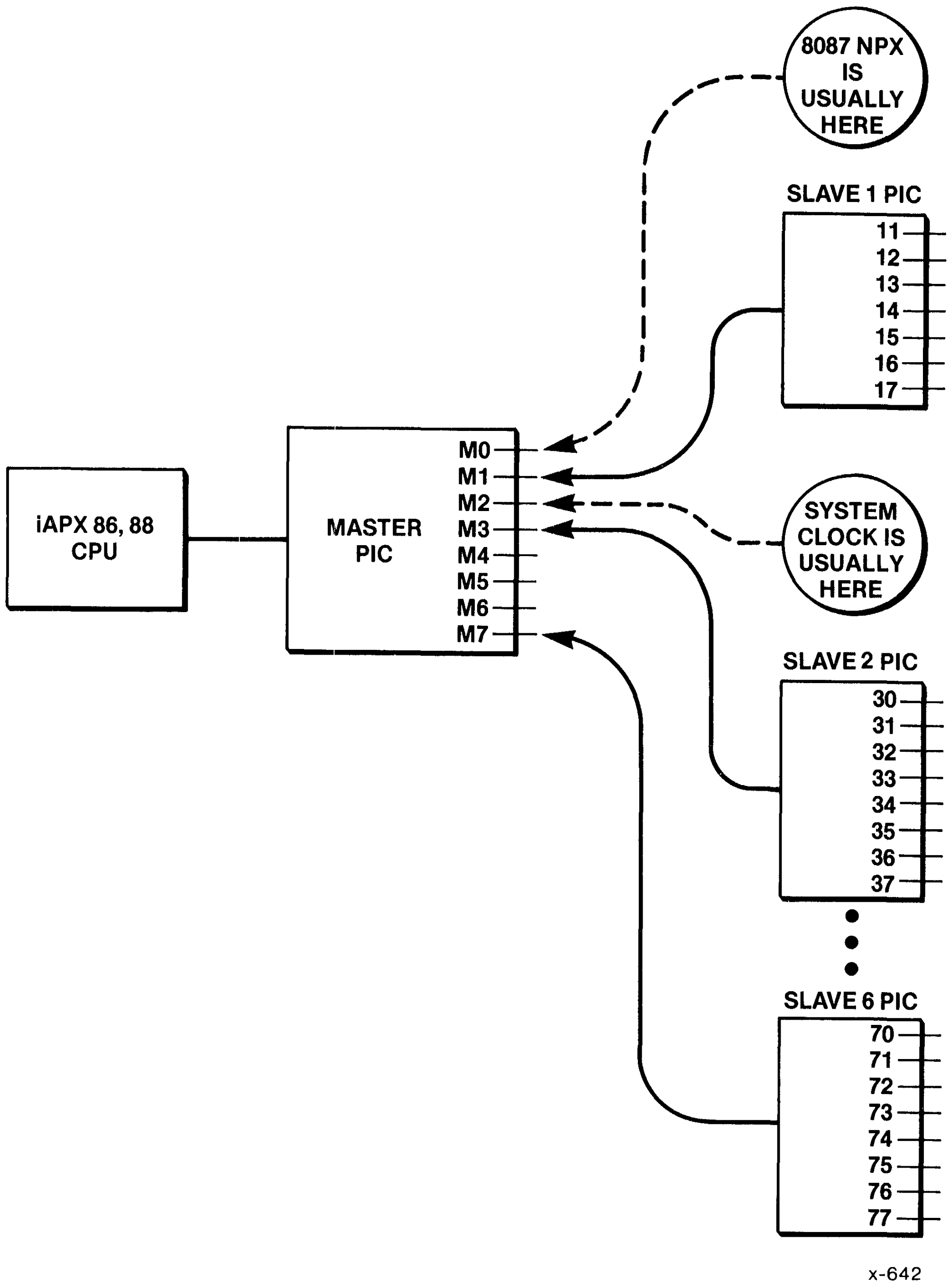


Figure 8-1. iAPX 86, 88 Interrupt Lines

iAPX 286 Configurations

An iAPX 286 environment is similar to an iAPX 86, 88 environment in all ways but one. If your iAPX 286-based system includes an 80287 NPX, you do not have to connect the NPX to a PIC. Instead of using the PIC, the NPX uses CPU interrupt traps 7 and 16 to communicate directly with the iAPX 286 component. This setup results in an extra interrupt level for you to use any way you wish. Figure 8-2 illustrates this situation.

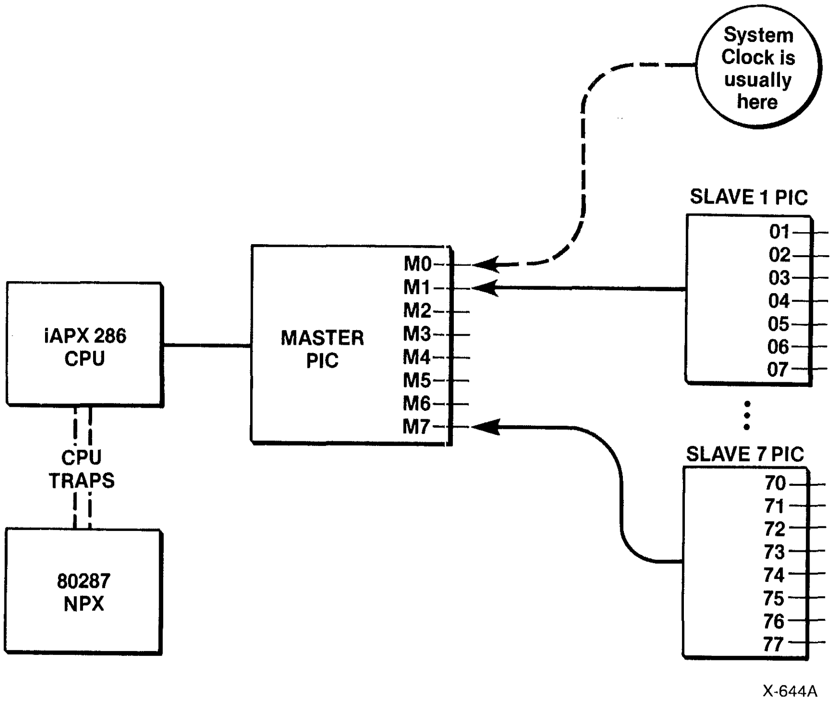


Figure 8-2. iAPX 286 Interrupt Lines

```
/***********************************************************************
 *  The calling task invokes the SEND$UNITS system call to send the    *
 *  units to the semaphore just created (sem$token.)                    *
 **********************************************************************/

    CALL RQ$SEND$UNITS              (sem$token,
                                     three$units$sent,
                                     @status);
         •
         •      Typical PL/M-86 Statements
         •

END SAMPLE_PROCEDURE;
```

CONDITION CODES

E$OK — No exceptional conditions.

E$EXIST — The semaphore parameter is not a token for an existing object.

E$LIMIT — The number of units that the calling task is trying to send would cause the semaphore's supply of units to exceed its maximum allowable supply.

E$NOT$CONFIGURED — This system call is not part of the present configuration.

E$TYPE — The semaphore parameter is a token for an object that is not a semaphore.

SET$EXCEPTION$HANDLER

SET$EXCEPTION$HANDLER assigns an exception handler to the calling task.

```
    CALL RQ$SET$EXCEPTION$HANDLER (exception$info$ptr, except$ptr);
```

INPUT PARAMETER

exception$info$ptr — A POINTER to a structure of the following form:

```
STRUCTURE(
   EXCEPTION$HANDLER$OFFSET    WORD,
   EXCEPTION$HANDLER$BASE      SELECTOR,
   EXCEPTION$MODE              BYTE);
```

where:

- exception$handler$offset contains the offset of the first instruction of the exception handler.
- exception$handler$base contains the base of the CPU segment containing the first instruction of the exception handler.
- exception$mode contains an encoded indication of the calling task's intended exception mode. The value is interpreted as follows:

Value	When to Pass Control to Exception Handler
0	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

If exception$handler$offset and exception$handler$base both contain zeros, the exception handler of the calling task's parent job is assigned.

OUTPUT PARAMETER

except$ptr — A POINTER to a WORD to which the iRMX 86 Operating System will return the condition code generated by this system call.

SIGNAL$EXCEPTION

The SIGNAL$EXCEPTION system call is invoked by OS extensions to signal the occurrence of an exceptional condition.

```
    CALL RQ$SIGNAL$EXCEPTION(exception$code, param$num, stack$pointer,
                             reserved, npx$status$word, except$ptr);
```

INPUT PARAMETERS

exception$code — A WORD containing the code (see list in Chapter 7) for the exceptional condition detected.

param$num — A BYTE containing the number of the parameter which caused the exceptional condition. If no parameter is at fault, param$num equals zero.

stack$pointer — A WORD which, if not zero, must contain the value of the stack pointer saved on entry to the operating system extension (see the entry procedure in Chapter 10 for an example). The top five words in the stack (where BP is at the top of the stack) must be as follows:

FLAGS	Saved by software interrupt
CS	to OS extension
IP	
DS	Saved by OS extension
BP	on entry

Upon completion of SIGNAL$EXCEPTION, control is returned to either of two instructions. If stack$pointer contains a zero, control returns to the instruction following the call to SIGNAL$EXCEPTION. Otherwise, control returns to the instruction identified in CS and IP.

reserved — A WORD reserved for Intel use. Set this parameter to zero.

npx$status$word — A WORD containing the status of the NPX.

OUTPUT PARAMETER

except$ptr — A POINTER to a WORD to which the iRMX 86 Operating System will return the condition code generated by this system call.

DESCRIPTION

Operating system extensions use the SIGNAL$EXCEPTION system call to signal the occurrence of exceptional conditions. Depending on the exceptional condition and the calling task's exception mode, control may or may not pass directly to the task's exception handler.

If the exception handler does not get control, the exceptional condition code is returned to the calling task. The task can then access the code by checking the contents of the word pointed to by the except$ptr parameter <u>for its call</u> (not for the call to SIGNAL$EXCEPTION).

EXAMPLE

```
/************************************************************************
 *  This example illustrates how the SIGNAL$EXCEPTION system call can   *
 *  be used to signal the occurrence of the exceptional condition       *
 *  E$CONTEXT.                                                          *
 ************************************************************************/

    $INCLUDE(:F1:SAMPLE.EXT);      /* Declares all system calls */

    DECLARE e$context              LITERALLY '5H';
    DECLARE param$num              BYTE;
    DECLARE stack$pointer          WORD;
    DECLARE reserved$word          LITERALLY '0';
    DECLARE status                 WORD;

SAMPLE_PROCEDURE:
    PROCEDURE;

    param$num = 0;                 /* no parameter at fault */
    stack$pointer = 0;             /* return control to instruction
                                      following call */
        •
        •     Typical PL/M-86 Statements
        •
```

EXAMPLE

```
/***********************************************************************
 *  This example illustrates how the SIGNAL$INTERRUPT system call can   *
 *  be used to activate an interrupt task.                               *
 ***********************************************************************/

    $INCLUDE(:F1:SAMPLE.EXT);      /* Declares all system calls */

    DECLARE the$first$word         WORD;
    DECLARE interrupt$level$7      LITERALLY '0000 0000 0111 1000B';
            /* specifies master interrupt level 7 */
    DECLARE interrupt$task$flag    BYTE;
    DECLARE interrupt$handler      POINTER;
    DECLARE data$segment           WORD;
    DECLARE status                 WORD;
    DECLARE interrupt$status       WORD;
    DECLARE ds$pointer             POINTER;
    DECLARE PTR$OVERLAY            LITERALLY 'STRUCTURE (offset   WORD,
                                                         base     WORD)';
                                   /* establishes a structure for
                                      overlays */
    DECLARE ds$pointer$ovly        PTR$OVERLAY AT (@ds$pointer);
                                   /* using the overlay structure, the
                                      base address of the interrupt
                                      handler's data segment is
                                      identified */

INTERRUPT_HANDLER: PROCEDURE INTERRUPT 59 PUBLIC;     /* 59 is meaningless
                                                      value.  ENTER$INTER-
                                                      RUPT establishes
                                                      actual level */

            •
            •     Typical PL/M-86 Statements
            •

/***********************************************************************
 *  The calling interrupt handler invokes the ENTER$INTERRUPT system    *
 *  call which loads a base address value (defined by                   *
 *  ds$pointer$ovly.base) into the data segment register.  This         *
 *  register provides a mechanism for the interrupt handler to pass     *
 *  data to the interrupt task to be started up by the SIGNAL$INTERRUPT *
 *  system call.                                                         *
 ***********************************************************************/

    CALL RQ$ENTER$INTERRUPT        (interrupt$level$7,
                                   @interrupt$status);
    CALL INLINE_ERROR_PROCESS      (interrupt$status);

            •
            •     Typical PL/M-86 Statements
            •
```

```
/***************************************************************************
 *  The interrupt handler uses SIGNAL$INTERRUPT to start up its           *
 *  associated interrupt task.                                            *
 **************************************************************************/
    CALL RQ$SIGNAL$INTERRUPT       (interrupt$level$7,
                                   @interrupt$status);
    CALL INLINE_ERROR_PROCESS      (interrupt$status);

END INTERRUPT_HANDLER;

INLINE_ERROR_PROCESS: PROCEDURE;
    IF interrupt$status <> E$OK THEN
         DO;
             •
             •     In-line Error Processing PL/M-86 Statements
             •
         END;

END INLINE_ERROR_PROCESS;

SAMPLE_PROCEDURE:
    PROCEDURE;

    ds$pointer = @the$first$word; /* a dummy identifier used to point to
                                      interrupt handler's data segment */
    data$segment = ds$pointer$ovly.base;
                                   /* identifies the base address of the
                                      interrupt handler's data segment */
    interrupt$handler = INTERRUPT$PTR (INTERRUPT_HANDLER);
                                   /* points to the first instruction of
                                      the interrupt handler */
    interrupt$task$flag = 01H;     /* indicates that calling task is to be
                                      interrupt task */
         •
         •     Typical PL/M-86 Statements
         •

/***************************************************************************
 *  By first invoking the SET$INTERRUPT system call, the calling task     *
 *  sets up an interrupt level and becomes the interrupted task for       *
 *  level 7.                                                              *
 **************************************************************************/

    CALL RQ$SET$INTERRUPT          (interrupt$level$7,
                                   interrupt$task$flag,
                                   interrupt$handler,
                                   data$segment,
                                   @status);
         •
         •     Typical PL/M-86 Statements
         •

END SAMPLE_PROCEDURE;
```

Note that, whether you specify a hard detach or not, there will be no attached files on the device after the device is detached.

CONDITION CODES

A$PHYSICAL$DETACH$DEVICE can return condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential code. A code returned as a result of asynchronous processing is a concurrent exception code. A complete explanation of sequential and concurrent parts of system calls is in Chapter 7 of this manual.

The following list is divided into two parts -- one for sequential codes and one for concurrent codes.

Sequential Condition Codes

The Basic I/O System can return the following exception codes to the word specified by the except$ptr parameter of this system call.

E$OK — No exceptional conditions.

E$EXIST — One or more of the following parameters is not a token for an existing object:

- The connection parameter
- The resp$mbox parameter

E$LIMIT — The calling task's job has already reached its object limit.

E$MEM — The memory available to the calling task's job is not sufficient to complete the call.

E$NOT$CONFIGURED — This system call is not part of the present configuration.

E$NOT$DEVICE$CONN — The specified connection parameter is not a device connection.

E$SUPPORT — The specified connection was not created by this job.

E$TYPE — At least one of the following is true:

- The connection parameter is a token for an object that is not a connection.
- The resp$mbox parameter is a token for an object that is not a mailbox.

Concurrent Exception Codes

The Basic I/O System will return the following codes in an I/O result segment at the mailbox specified by resp$mbox. After examining the segment, you should delete it.

E$OK	No exceptional conditions.
E$FNEXIST	The device specified by the connection parameter is already being detached.
E$IO	An I/O error occurred during the operation, but the operation was successful anyway.
E$OUTSTANDING$-CONNS	The call attempted a soft detach, but connections to the device still existed.

Bits   Value and Meaning

0 = Coordinates increase from left to right.

1 = Coordinates decrease from left to right.

12   Vertical axis orientation control (corresponds to OSC characters T:F). This specifies whether the coordinates on the terminal's vertical axis increase or decrease as you move from top to bottom across the screen.

0 = Coordinates increase from top to bottom.

1 = Coordinates decrease from top to bottom.

13-15   Reserved bits. For future compatibility, set to 0.

NOTE

If bits 4-5 contain 2 or 3, and bits 6-8 also contain 2 or 3, then they must both contain the same value. That is, they must both reflect the same parity convention (even or odd).

in$baud$rate   The input baud rate indicator (corresponds to OSC characters T:I). If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The word is encoded as follows:

0 = Invalid.

1 = Perform an automatic baud rate search.

Other = Actual input baud rate, such as 9600.

out$baud$rate   The output baud rate indicator (corresponds to OSC characters T:O). If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The word is encoded as follows:

0 = Invalid.

1 = Use the input baud rate for output.

Other = Actual output baud rate, such as 9600.

Most applications require the input and output baud rates to be equal. In such cases, use in$baud$rate to set the baud rate and specify a one for out$baud$rate.

scroll$lines — An operator at a terminal can enter a control character (default is Control-W) when he/she is ready for data to appear on the terminal's display screen. The scroll$lines value (corresponding to OSC characters T:S) specifies the maximum number of lines that are to be sent to the terminal each time the operator enters the control character. If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value.

x$y$size — The low-order byte of this word specifies the number of character positions on each line of the terminal's screen (and corresponds to OSC characters T:X). The high-order byte specifies the number of lines on the terminal's screen (and corresponds to OSC characters T:Y).

x$y$offset — The low-order byte of this word specifies the value that starts the numbering sequence of both the X and Y axes (and corresponds to OSC characters T:U). The high-order byte specifies the value to which the numbering of the axes must "fall back" after reaching 127 (and corresponds to OSC characters T:V).

The remaining fields apply only for intelligent communications boards (such as the iSBC 544 board) that maintain their own input and output buffers separately from the ones managed by the Basic I/O System's Terminal Support Code. If you aren't sure whether you can set these fields, invoke A$SPECIAL with function code 4 to get the terminal attributes. If bit 15 of the flow$control field (the next one described) is set, your board is a buffered device and you can set the following fields. (If your board is not a buffered device, setting any of the following fields will cause the terminal support code to return an E$PARAM Condition Code.)

flow$control — Specifies whether the communications board sends flow control characters (selected by the fc$on$char and fc$off$char fields, but usually XON and XOFF) to turn input on and off (corresponds to the OSC characters T:G). The low-order bit (bit 0) controls this option, as follows:

0	Disable flow control.
1	Enable flow control.

```
    DELCLARE  read$file$mark    STRUCTURE(
       search          BYTE);
```

where:

search — A value indicating the direction of the search, as follows:

00	Search forward
OFFH	Search backward (for start/stop drives only)

When your task issues the A$SPECIAL system call with spec$func set to 9, the tape drive writes a file mark at the current position on the tape. This function also terminates tape write operations.

When your task issues the A$SPECIAL system call with spec$func set to 10, the tape drive fast-forwards the tape to the end and then rewinds it to the load point.

CONDITION CODES

A$SPECIAL return condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in Chapter 7 of this manual.

The following list is divided into two parts -- one for sequential codes, and one for concurrent codes.

Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

E$OK — No exceptional conditions.

E$BUFFERED$CONN — The connection parameter is a connection produced by the Extended I/O System. You cannot use it with Basic I/O System calls.

E$EXIST — At least one of the following is true:

- One or more of the following parameters or fields is not a token for an existing object:

  - The connection parameter
  - The resp$mbox parameter
  - The mailbox field in the notify structure. (Spec$func = 2.)
  - The object field in the notify structure. (Spec$func = 2.)
  - The semaphore field in the signal$pair structure. (Spec$func = 6.)

- The connection is being deleted.

E$IFDR — The function requested (spec$func) is not valid for the type of file specified by the connection parameter.

E$LIMIT — The calling task's job has already reached its object limit.

E$MEM — The memory available to the calling task's job is not sufficient to complete the call.

E$NOT$CONFIGURED — This system call is not part of the present configuration.

E$PARAM — At least one of the following is true:

- The spec$func parameter was 5, and one or more of the following is true:
  - Bits 0-1 of the connection$flags field was equal to 0.
  - Bits 6-8 of the terminal$flags field was greater than 4.
- The spec$func parameter was 6, and the character field was greater than 1FH.
- The spec$func parameter was greater than 10.
- One or more of the fields related to buffered devices (high$water$mark, low$water$mark, fc$on$char, fc$off$char) was set while bit 15 of the flow$control field was reset to zero (specifying an un-buffered device).

E$SUPPORT — The specified connection was not created by this job.

E$TYPE — One or more of the following parameters or fields is a token for an existing object of the wrong type:

- The connection parameter.
- The resp$mbox parameter.
- The mailbox field of the notify structure. (Spec$func = 2.)
- The semaphore field of the signal$pair structure. (Spec$func = 6.)

Concurrent Condition Codes

The Basic I/O System can return the following condition codes in an I/O result segment at the mailbox specified by resp$mbox. After examining the segment, you should delete it.

E$OK — No exceptional conditions.

E$CONN$NOT$OPEN — The specified connection is not open. This applies only to stream and physical files.

E$FLUSHING — The specified connection was closed before the function could be completed.

E$IDDR The specified function is not supported by the device containing the file.

E$IFDR — The connection refers to a named file, but the function is not "notify".

E$IO — An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the I/O result segment for more information.

E$NOT$DEVICE$CONN — The function code is 'notify', but the specified connection is not a device connection. This applies only to named and physical files.

E$SPACE — One of the following is true:

- This call attempted to format a track of a physical file that is beyond the end of the volume.
- This call attempted to format a track of a RAM disk other than track 0.

E$STREAM$SPECIAL One of the following is true:

- This is a "query" request, but another query is already queued. This applies only to stream files.
- This is a "satisfy" request, but either a query request is queued, or no requests are queued. This applies only to stream files. (See Artificially Satisfying a Stream File I/O Request in the DESCRIPTION.)

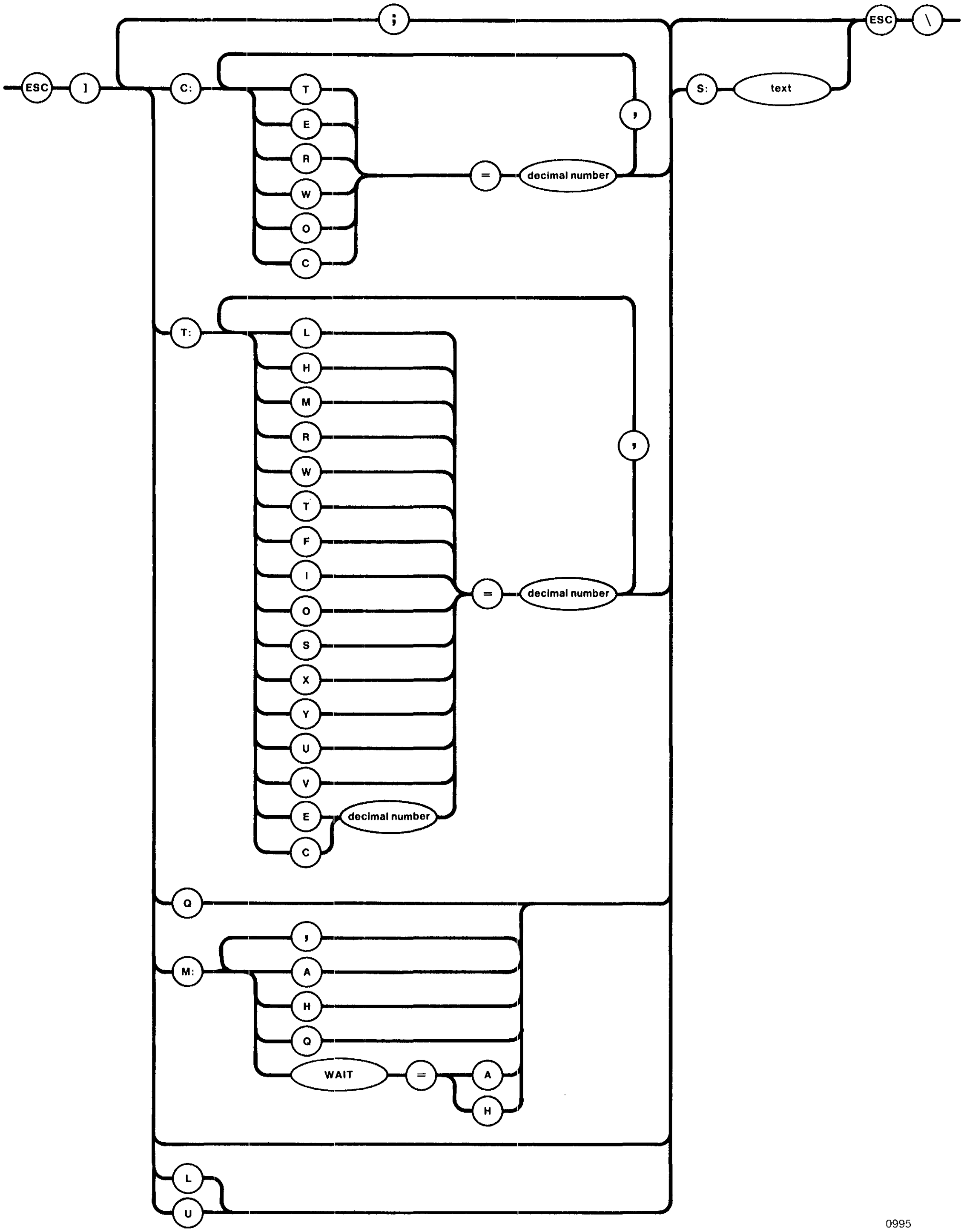


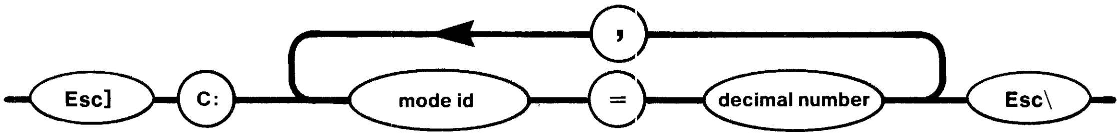
Figure F-1. Composite OSC Sequence Diagram

MODES THAT A TERMINAL INHERITS FROM A CONNECTION

This section describes the modes that depend on the connection to the terminal, rather than on the terminal itself. With these modes, when multiple connections to a terminal exist, the terminal might operate one way when communicating via the first connection and a different way when communicating via the second connection.

Each of these modes relates directly to one or more bits in the connection$flags word for the connection (as defined in the Chapter 8 description of the A$SPECIAL system call). The names of the modes, the single-letter identification codes for the modes, the bits of the connection$flags word to which they correspond, and a brief description of their functions are given in Table F-2.

Assuming that the OSC control mode is set appropriately, the modes that a terminal inherits from a connection can be altered. The syntax of an OSC sequence that will change one or more of these modes is as follows:



0997

where:

C: Indicates that this sequence applies to a connection. The Terminal Support Code ignores all but the first letter, so you can supply any group of characters that begins with "C". However, you must include the colon (:) at the end.

mode id An ID letter from the list of modes given in Table F-2.

decimal number The value to which you want to set the mode. This number must be of the character data type.

Table F-2 contains a brief description of the modes and values. For a more complete description, refer to the description of A$SPECIAL in Chapter 8.

iRMX™ 86 Release 6.Ø Change Package: Update 2

Change Pages for:

iRMX™ 86 Programmer's Reference Manual, Part II (146196-ØØ1)

C$SEND$CO$RESPONSE

C$SEND$CO$RESPONSE, a message processing call, sends a message to :CO: and reads a response from :CI:.

```
    CALL RQ$C$SEND$CO$RESPONSE(response$p, response$max, message$p,
                               except$ptr);
```

INPUT PARAMETERS

message$p — A POINTER to a STRING containing the message to be sent to :CO:. If zero, no message is sent.

response$max — A WORD whose value specifies the length in bytes of the string pointed to by the response$p parameter. The value in response$max must equal the length of the string plus one (stringlength + 1). If response$max is zero or one, no response from :CI: will be requested; control returns to the calling task immediately.

OUTPUT PARAMETERS

response$p — A POINTER to a STRING that receives the operator's response from :CI:.

except$ptr — A POINTER to a WORD in which the Human Interface returns a condition code.

DESCRIPTION

When used with all its features, C$SEND$CO$RESPONSE sends the string pointed to by message$p to :CO: and waits for a response from :CI:. It places this response in the string pointed to by response$p. However, If message$p is zero, C$SEND$CO$RESPONSE omits sending the message to :CO:; if either response$max or response$p is zero, it does not wait for a response from :CI:. Therefore, the operations performed by C$SEND$CO$RESPONSE depend on the values of the message$p and response$max parameters, as follows:

message$p	response$max	Action
zero	zero	Perform no I/O
zero	non-zero	Send no message, wait for input
non-zero	non-zero	Send message, wait for input
non-zero	zero	Send message, don't wait

If C$SEND$CO$RESPONSE requests a response from :CI:, output from other tasks can be displayed at :CO: while the system waits for a response from :CI:.

The main distinction between C$SEND$CO$RESPONSE and C$SEND$EO$RESPONSE calls is that C$SEND$EO$RESPONSE always sends messages to and receives messages from the operator's terminal; input and output cannot be redirected to another device. In contrast, C$SEND$CO$RESPONSE sends messages to :CO: and receives messages from :CI:; therefore, programs such as SUBMIT can redirect this input and output.

EXCEPTION CODES

E$OK — No exceptional conditions were encountered.

E$CONTEXT — The calling task's job was not created by the Human Interface.

E$CONNECTION$-OPEN — At least one of the following is true:

- The connection to :CI: was not open for reading or the connection to :CO: was not open for writing.
- The connection to :CI: or :CO: was not open.
- The connection to :CI: or :CO: was opened with A$OPEN rather than S$OPEN.

E$EXIST — The token value for :CI: or :CO: is not a token for an existing object.

E$FLUSHING — The device containing the :CI: and :CO: files was being detached.

E$IO$HARD — While attempting to access the :CI: or :CO: file, the Operating System detected a hard I/O error.

E$IO$OPRINT — While attempting to access the :CI: or :CO: file, this call found that the device was off-line. Operator intervention is required. C$FORMAT$EXCEPTION returns the value E$IO$NOT$READY for this code.

E$IO$SOFT While attempting to access the :CI: or :CO: file, this call detected a soft I/O error. It tried again, but was unsuccessful. Another try might be successful.

E$IO$UNCLASS An unknown type of I/O error occurred while this call tried to access the :CI: or :CO: file.

E$IO$WRPROT While attempting to obtain a connection to the :CO: file, this call found that the volume containing the file is write-protected.

E$LIMIT At least one of the following is true:

- The calling task's job has already reached its object limit.
- The calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations.
- The calling task's job was not created by the Human Interface.

E$MEM The memory available to the calling task's job is not sufficient to complete the call.

E$NOT$CONNECTION The call obtained a token for an object that should have been a connection to :CI: or :CO: but was not a file connection.

E$PARAM The call attempted to write beyond the end of a physical file.

E$SPACE One of the following is true:

- The output volume is full.
- The call attempted to write beyond the end of a physical file.

E$STREAM$SPECIAL When attempting to read or write to :CI: or :CO:, the Extended I/O System issued an invalid stream file request.

E$SUPPORT The connection to :CI: or :CO: was not created by this job.

E$TIME The calling task's job was not created by the Human Interface.

C$SEND$EO$RESPONSE

C$SEND$EO$RESPONSE, a message processing call, sends a message to and reads a response from the operator's terminal.

```
CALL RQ$C$SEND$EO$RESPONSE(response$p, response$max, message$p,
                           except$ptr);
```

INPUT PARAMETERS

message$p — A POINTER to a STRING containing the message to be sent to the operator's terminal. If zero, no message is sent.

response$max — A WORD whose value specifies the length in bytes of the string pointed to by the response$p parameter. The value in response$max must equal the length of the string plus one (stringlength + 1). If response$max is zero or one, no response from the operator's terminal will be requested; control returns to the calling task immediately.

OUTPUT PARAMETERS

response$p — A POINTER to a STRING that receives the operator's response from the terminal.

except$ptr — A POINTER to a WORD in which the Human Interface returns a condition code.

DESCRIPTION

When used with all its features, C$SEND$EO$RESPONSE sends the string pointed to by message$p to the operator's terminal and waits for a response from the operator. It places this response in the string pointed to by response$p. However, if message$p is zero, C$SEND$EO$RESPONSE omits sending the message to the operator; if either response$max or response$p is zero, it does not wait for a response. Therefore, the operations performed by C$SEND$EO$RESPONSE depend on the values of the message$p and response$max parameters, as follows:

message$p	response$max	Action
zero	zero	Perform no I/O
zero	non-zero	Send no message, wait for input
non-zero	non-zero	Send message, wait for input
non-zero	zero	Send message, don't wait

DQ$RENAME

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

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

INPUT PARAMETERS

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

OUTPUT PARAMETER

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

DESCRIPTION

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

A file's pathname may be changed in any way, provided that the file or directory remains on the same volume.

DQ$RESERVE$IO$MEMORY

The DQ$RESERVE$IO$MEMORY lets your program reserve enough memory to ensure that it can open and attach the files it will be using.

```
    CALL DQ$RESERVE$IO$MEMORY (number$files, number$buffers, except$ptr);
```

INPUT PARAMETERS

number$files — A WORD whose value indicates the maximum number of files the program will have attached simultaneously. This value must not be greater than 12. Moreover, no more than 6 of these files may be open simultaneously.

number$buffers — A WORD whose value indicates the total number of buffers (up to a maximum of 12) that will be needed at one time. For example, if your program will have two files open at the same time, and each of them has two buffers (specified when they are opened), number$files should be two and number$buffers four.

OUTPUT PARAMETER

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

DESCRIPTION

DQ$RESERVE$IO$MEMORY sets aside memory on behalf of the calling program, guaranteeing that it will be available when needed later for attaching and opening files. This memory is used for internal UDI data structures when the program requests file connections via DQ$ATTACH and for buffers when the program opens file connections via DQ$OPEN. Memory reserved in this way is not eligible to be allocated by DQ$ALLOCATE. Your program should call DQ$RESERVE$IO$MEMORY before making any calls to DQ$ALLOCATE.

In the call to DQ$RESERVE$IO$MEMORY, you may specify as many as 12 files (that can be attached using the reserved memory) and as many as 12 buffers (that can be requested when opening files).

The format of the call to the Finish I/O procedure is as follows:

```
CALL finish$io(duib$p, ddata$t);
```

where:

finish$io — Name of the Finish I/O procedure. You can specify any name for this procedure as long as it does not conflict with other procedure names. You must, however, provide its starting address in the DUIBs of all device-units that it services.

duib$p — POINTER to the DUIB of the device-unit of the device being detached. The finish$io procedure needs this DUIB in order to determine the device on which to perform the final processing.

ddata$t — SELECTOR containing the location of the data storage area originally created by the init$io procedure. The finish$io procedure must delete this resource and any others created by driver routines.

QUEUE I/O PROCEDURE

The I/O System calls the Queue I/O procedure to place an I/O request on a queue, so that it can be processed when the device is not busy. The Queue I/O procedure must actually start the processing of the next I/O request on the queue if the device is not busy. The format of the call to the Queue I/O procedure is as follows:

```
CALL queue$io(iors$t, duib$p, ddata$t);
```

where:

queue$io — Name of the Queue I/O procedure. You can use any name for this procedure as long as it does not conflict with other procedure names. You must, however, provide its starting address for the DUIBs of all device-units that it services.

iors$t — SELECTOR containing the location of an IORS. This IORS describes the request. When the request is processed, the driver (though not necessarily the queue$io procedure) must fill in the status fields and send the IORS to the response mailbox (exchange) indicated in the IORS. Chapter 2 describes the format of the IORS. It lists the information that the I/O System supplies when it passes the IORS to the queue$io procedure and indicates the fields of the IORS that the device driver must fill in.

duib$p — POINTER to the DUIB of the device-unit for which the request is intended.

ddata$t — SELECTOR containing the location of the data storage area originally created by the init$io procedure. The queue$io procedure can place any necessary information in this area in order to update the request queue or status fields.

CANCEL I/O PROCEDURE

The I/O System can call the Cancel I/O procedure in order to cancel one or more previously queued I/O requests. The iRMX 88 I/O System does not call Cancel I/O, but in the iRMX 86 environment Cancel I/O is called under either of the following two conditions:

- If the user makes an RQ$A$PHYSICAL$DETACH$DEVICE system call and specifies the hard detach option (refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for a description of this call). This system call forcibly detaches all objects associated with a device-unit.
- If the job containing the task which made an I/O request is deleted. The I/O System calls the Cancel I/O procedure to remove any requests that tasks in the deleted job might have made.
- If the user deletes a connection to a device. The I/O system calls Cancel I/O to remove any I/O requests pending for that device.

If the device cannot guarantee that a request will be finished within a fixed amount of time (such as waiting for input from a terminal keyboard), the Cancel I/O procedure must actually stop the device from processing the request. If the device guarantees that all requests finish in an acceptable amount of time, the Cancel I/O procedure does not have to stop the device itself, but only removes requests from the queue.

The format of the call to the Cancel I/O procedure is as follows:

```
CALL cancel$io(cancel$id, duib$p, ddata$t);
```

where:

cancel$id — Name of the Cancel I/O procedure. You can use any name for this procedure as long as it doesn't conflict with other procedure names. You must, however, provide its starting address in the DUIBs of all device-units that it services.

cancel$id — WORD containing the id value for the I/O requests that are are to be cancelled. Any pending requests with this value in the cancel$id field of their IORS's must be removed from the queue of requests by the Cancel I/O procedure. Moreover, the I/O System places a CLOSE request with the same cancel$id value in the queue. The CLOSE request must not be processed until all other requests with that cancel$id value have been returned to the I/O System.

duib$p — POINTER to the DUIB of the device-unit for which the request cancellation is intended.

ddata$t — SELECTOR containing the location of the data storage area originally created by the init$io procedure. This area may contain the request queue.

IMPLEMENTING A REQUEST QUEUE

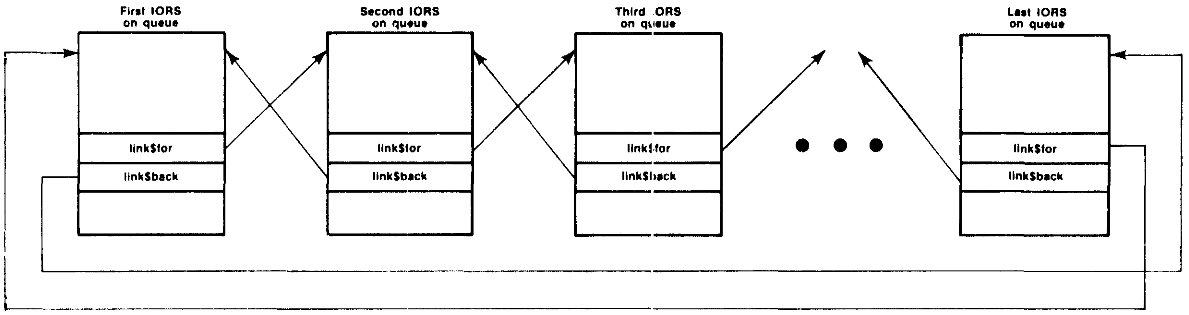
Making I/O requests via system calls and the actual processing of these requests by I/O devices are asynchronous activities. When a device is processing one request, many more can be accumulating. Unless the device driver has a mechanism for placing I/O requests on a queue of some sort, these requests will become lost. The common and random access device drivers form this queue by creating a doubly linked list. The list is used by the QUEUE$IO and CANCEL$IO procedures, as well as by INTERRUPT$TASK.

Using this mechanism of the doubly linked list, common and random access device drivers implement a FIFO queue for I/O requests. If you are writing a custom device driver, you might want to take advantage of the LINK$FOR and LINK$BACK fields that are provided in the IORS and implement a scheme similar to the following for queuing I/O requests.

Each time a user makes an I/O request, the I/O System passes an IORS for this request to the device driver, in particular to the Queue I/O procedure of the device driver. The common and random access driver Queue I/O procedures make use of the LINK$FOR and LINK$BACK fields of the IORS to link this IORS together with IORSs for other requests that have not yet been processed.

This queue is set up in the following manner. The device driver routine that is actually sending data to the controller accesses the first IORS on the queue. The LINK$FOR field in this IORS points to the next IORS on the queue. The LINK$FOR field in the second IORS points to the third IORS on the queue, and so forth until, in the last IORS on the queue, the LINK$FOR field points back to the first IORS on the queue. The LINK$BACK fields operate in the same manner. The LINK$BACK field of the last IORS on the queue points to the previous IORS. The LINK$BACK field of the second to last IORS points to the third to last IORS on the queue, and so forth, until, in the first IORS on the queue, the LINK$BACK field points back to the last IORS in the queue. A queue of this sort is illustrated in Figure 6-1.

The device driver can add or remove requests from the queue by adjusting LINK$FOR and LINK$BACK pointers in the IORSs.



x-679

Figure 6-1. Request Queue

To handle the dual problems of locating the queue and ascertaining whether the queue is empty, you can use a variable such as head$queue. If the queue is empty, head$queue contains the value 0. Otherwise, head$queue contains the address of the first IORS in the queue.

***

```
$subtitle('Initialize Time')
/************************************************************************
*   init_time                                                           *
*                                                                       *
*     This procedure zeros the timer, creates a task to                 *
*     maintain the timer, and a region to ensure exclusive              *
*     access to the timer.  This procedure must be called               *
*     before the first time that get_time or set_time is                *
*     called.  Also, this procedure should be called only               *
*     once.  The easiest way to make sure this happens is to            *
*     call init_time from your initialization task.                     *
*                                                                       *
*     The timer task will run in the job from which this                *
*     procedure is called.                                              *
*                                                                       *
*     If your application experiences a lot of interrupts,              *
*     the timer may run slow.  You can rectify this                     *
*     problem by raising the priority of the timer                      *
*     task.  To do this, change the 128 in the                          *
*     rq$create$task system call to a smaller number.                   *
*     This change may slow the processing of your                       *
*     interrupts.                                                       *
************************************************************************/

init_time: PROCEDURE(ret_status_p) REENTRANT PUBLIC;

    DECLARE ret_status_p       POINTER,
            ret_status         BASED ret_status_p WORD,
            timer_task_t       TASK,
            local_status       WORD;

    time_in_sec = 0;

    time_region = rq$create$region      /* Create a region. */
                    (PRIORITY_QUEUE, ret_status_p);

    IF (ret_status <> E$OK) THEN
        RETURN;                         /* Return with error. */

    data_seg_p = @data_seg_p;           /* Get contents of
                                           DS register. */

    timer_task_t = rq$create$task       /* Create timer task. */
                   (128,                /*  priority         */
                    @maintain_time,     /*  start addr       */
                    data_seg_p_o.base,  /*  data seg base    */
                    0,                  /*  stack ptr        */
                    512,                /*  stack size       */
                    0,                  /*  task flags       */
                    ret_status_p);
    IF (ret_status <> E$OK) THEN
      CALL rq$delete$region                   /* Since could not */
           (time_region, @local_status);  /* create task,    */
                                              /* must delete     */
                                              /* region.         */

END init_time;

END timer;
```

***

CHAPTER 3
PROGRAMMING CONSIDERATIONS

The iRMX 86 Terminal Handler supports terminal input and output by providing mailbox interfaces. Figure 3-1 shows the use of these mailboxes. In the figure, an arrow pointing from a task to a mailbox represents an RQ$SEND$MESSAGE system call. An arrow pointing from a mailbox to a task indicates an RQ$RECEIVE$MESSAGE system call.

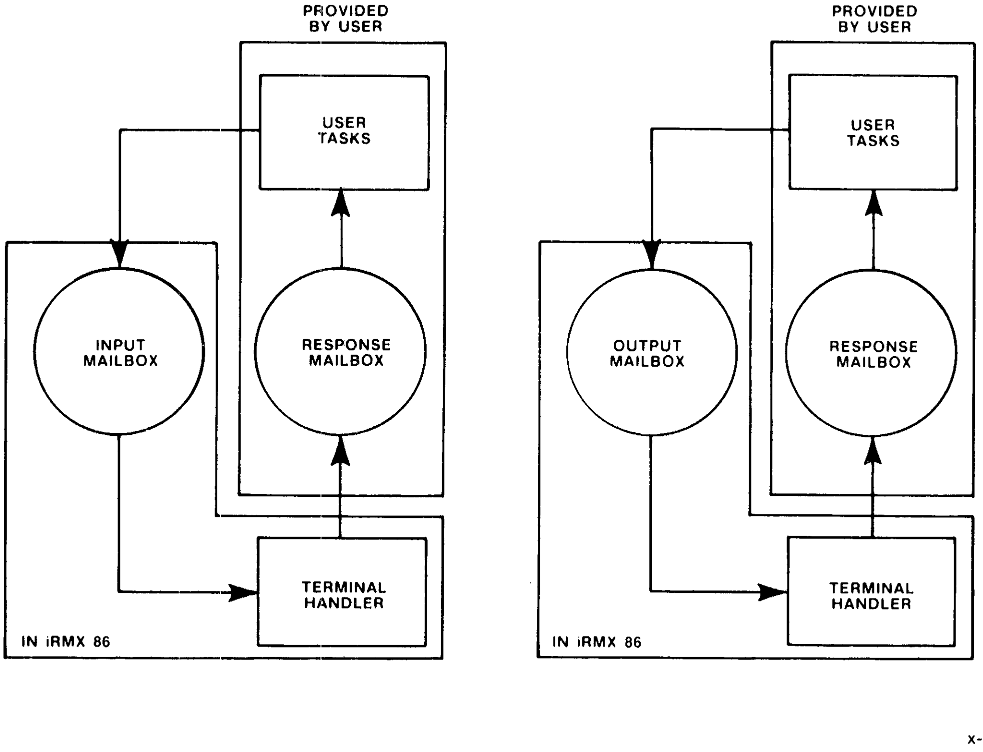


Figure 3-1. Input and Output Mailbox Interfaces

The protocol that tasks observe is much the same for input and output. In each case, the task initiates I/O by sending a request message to a mailbox. An input request mailbox (default name RQTHNORMIN) and an output request mailbox (default name RQTHNORMOUT) are provided. These mailboxes are cataloged in the root job directory. In the case of multiple terminals, one input and one output mailbox will be cataloged for each Terminal Handler. (See Chapter 4 for more information about multiple versions of the Terminal Handler.) Figure 3-2 illustrates the protocol for finding the root job token and for obtaining the input and output mailbox tokens.

```
/*************************************************************************
*  This example illustrates the protocol for finding the root job token *
*  and for obtaining the input and output mailbox tokens.               *
*************************************************************************

DECLARE rtjb$token                     WORD;
DECLARE root$job                       LITERALLY '3';
DECLARE status                         WORD;

DECLARE input$mbx$token                WORD;

DECLARE wait$forever                   LITERALLY 'OFFFFH';


/*By setting the input parameter to three, the GET$TASK$TOKEN primitive
  will return the root job's TOKEN.*/

rtjb$token = RQ$GET$TASK$TOKENS        (root$job,
                                        @status);

/*The following LOOKUP$OBJECT primitives use the default mailbox names.*/

input$mbx$token = RQ$LOOKUP$OBJECT     (rtjb$token,
                                        @(10, 'RQTHNORMIN'),
                                        wait$forever,
                                        @status);

output$mbx$token = RQ$LOOKUP$OBJECT    (rtjb$token,
                                        @(11, 'RQTHNORMOUT'),
                                        wait$forever,
                                        @status);
```

Figure 3-2. Protocol for Obtaining Root Job and Mailbox Tokens

Refer to the iRMX 86 NUCLEUS REFERENCE MANUAL for more information concerning the individual primitives used in the previous example. When a task sends a message to the Terminal Handler mailbox, the Terminal Handler processes the request and then sends a response message back to the requesting task. The task waits at a response mailbox for the message. Thus, whether a task does input or output, it first sends and then receives. The full details of the input and output protocols are described later in this chapter. Output is discussed first because it is somewhat easier to understand.

For both input and output, a task sends a message segment to the Terminal Handler. The format of a request message is depicted in Figure 3-2. The numbers in that figure are offsets, in bytes, from the beginning of the segment. The field names have different meanings for input and for output. For both input and output, the first four fields are WORD values. The MESSAGE CONTENT field can be up to 132 bytes in length for input and up to 65527 bytes in length for output.

```
;
; *-*-*   BS1.CSD   *-*-*
;
;   Generate the iAPX 86, 88 Bootstrap Loader V5.0 first stage.
;
;   Invocation:  submit bs1(first stage location, second stage location)
;
run
;
asm86 :f1:bs1.a86   macro(90) object(:f1:bs1.obj)   print(:f1:bs1.lst)
asm86 :f1:bserr.a86 macro(50) object(:f1:bserr.obj) print(:f1:bserr.lst)
asm86 :f1:b204.a86  macro(50) object(:f1:b204.obj)  print(:f1:b204.lst)
asm86 :f1:b206.a86  macro(50) object(:f1:b206.obj)  print(:f1:b206.lst)
asm86 :f1:b208.a86  macro(50) object(:f1:b208.obj)  print(:f1:b208.lst)
asm86 :f1:b215.a86  macro(50) object(:f1:b215.obj)  print(:f1:b215.lst)
asm86 :f1:b218a.a86 macro(50) object(:f1:b218.obj)  print(:f1:b218.lst)
asm86 :f1:b251.a86  macro(50) object(:f1:b251.obj)  print(:f1:b251.lst)
asm86 :f1:b254.a86  macro(50) object(:f1:b254.obj)  print(:f1:b254.lst)
asm86 :f1:bsasi.a86 macro(50) object(:f1:bsasi.obj) print(:f1:bsasi.lst)
asm86 :f1:bscsi.a86 macro(50) object(:f1:bscsi.obj) print(:f1:bscsi.lst)
;

link86
    :f1:bs1.obj,                      &
    :f1:bserr.obj,                    &
&   :f1:bcico.obj,                    & ;for stand-alone serial channel
                                      & ;support
    :f1:b204.obj,                     &
    :f1:b206.obj,                     &
    :f1:b208.obj,                     &
    :f1:b215.obj,                     &
    :f1:b218.obj,                     &
    :f1:b251.obj,                     &
    :f1:b254.obj,                     &
    :f1:bsasi.obj,                    &
    :f1:bscsi.obj,                    &
    :f1:bs1.lib                       &
    to :f1:bs1.lnk print(:f1:bs1.mp1) &
    nopublics except(first-stage,boot_186,bootstrap_entry)
;
loc86 :f1:bs1.lnk                                      &
    addresses(classes(code(%0),stack(%1)))             &
    order(classes(code,code_error,stack,data,boot))    &
    noinitcode                                         &
    start(first-stage)                                  &
 &  ; change above line to start(boot_186) if iAPX_186_INIT is invoked  &
    segsize(boot(1800H))                    &
    map print(:f1:bs1.mp2)                  &
    ; Add "bootstrap" to loc86 when locating the first stage in ROM
```

Figure 2-2. First Stage Configuration File BS1.CSD

```
;
exit
;
;   Bootstrap Loader first stage generation complete.
;
```

Figure 2-2. First Stage Configuration File BS1.CSD (continued)

counter_base_port — The 16-bit address of the base port used by the baud rate timer. This port varies according to the type of the device and, if applicable, the channel used on the device, as follows:

8253	Counter 0 Count Register Port
8254	Counter 0 Count Register Port
80130	ICW1 Register Port
80186	Use OFF00H on all Intel boards
82530 Channel A	Channel A Command Register Port
82530 Channel B	Channel B Command Register Port

counter_port_delta — The number of bytes between consecutive ports used by the timer.

baud_counter — The baud rate-generating counter on the timer. The devices and the counters you can specify for them are as follows:

8253	0, 1, and 2
8254	0, 1, and 2
80130	2
80186	0, 1
82530	0

count — A value that, when loaded into the timer register, generates the desired baud rate. The method of calculating this value is described in the paragraphs following these parameter definitions.

flags — A value that, when present, specifies which channel of an 82530 Serial Communications Controller will serve as your serial controller. If you give any value except 82530 for the serial_type parameter, omit this parameter; that is, write the macro as if the count parameter is the last parameter. If you give 82530 as the value of the serial_type parameter, specify A (for Channel A) or B (for Channel B) for this parameter.

iRMX™ 86 Release 6.Ø Change Package: Update 2

Change Pages for:

iRMX™ 86 Installation and Configuration Guide (146197-ØØ1)

Table 6-4. MULTIBUS® Priority Selection Jumpers (continued)

Intel Board	Remove Jumper	Add Jumper	Description/Function
iSBC 254S			Default setting selects serial priority.
	E130-131		Selects parallel priority.
iSBX 270			Not applicable.
iSBX 351			Not applicable.
iSBC 534			Not applicable.
iSBC 544			Not applicable.

5 1/4-INCH DRIVE SELECTION JUMPERS

Table 6-5 lists the jumpers you must change to incorporate a 5 1/4-inch flexible diskette into your system.

Table 6-5. 5 1/4-Inch Drive Selection Jumpers

Intel Board	Remove Jumper	Add Jumper	Description/Function
iSBC 204			Use of 5 1/4-inch diskette drives controlled by the iSBC 204 is not supported by iRMX 86.
iSBC 206			Not applicable.
iSBC 208	E18-E19	E4-E5 E6-E11 E17-E19	Selects 5 1/4-inch drives.

Table 6-5. 5 1/4-Inch Drive Selection Jumpers (continued)

Intel Board	Remove Jumper	Add Jumper	Description/Function
iSBC 208 (continued)	E21-E22	E17-E22	Install only if using double sided 5 1/4-inch diskette drives that do not supply a double-sided signal.
iSBC 215			Intel recommends that you use the iSBC 215G for controlling 5 1/4-inch winchester drives.
iSBC 215G	W1, 1-2	W1, 1-3	Selects 5 1/4-inch CMI, model 5412, winchester drives.
	W2, 1-2		
	W5, 1-2	W5, 1-3	
	W6, 1-2	W6, 1-3	
	W7, 1-2	W7, 1-3	
	W8, 1-2	W8, 1-3	
		W9, 1-2	
	W13, 1-3	W13, 1-2	
	W14, 1-3	W14, 1-2	
	W15, 1-2		
	W16, 1-3	W16, 1-2	
	W22, 1-3	W22, 1-2	
		W27, 1-2	
	W33, 1-3	W33, 1-2	
	W34, 1-2	W37, 1-2	
	W35, 1-2	W38, 1-2	
iSBX 218			Use of 5 1/4-inch diskette drives controlled by the iSBC 218 is not supported by iRMX 86.
iSBX 218A			Default configuration selects 5 1/4-inch diskette drives.
iSBC 220			Not applicable.
iSBX 251			Not applicable.

Table 6-7. Controller Board Switch Settings (continued)

Intel Board	Switch Setting	Description/Function
iSBC 220 (DIP switches)	S1, 1-7 OFF 8 ON	Selects port address 100H.
	S2, 1-2 ON	Selects a 16-bit bus and 16-bit address decoding.
	3-10 OFF	Selects port address 100H.
iSBC 220 (Wire Wraps)	E16 - E15	Selects port address 100H
	E18 - E17 E20 - E19	Selects a 16-bit bus and 16-bit address decoding
iSBX 251		Not applicable.
iSBC 254		Not applicable.
iSBC 254S		Not applicable.
iSBX 270		Not applicable.
iSBX 351		Not applicable.
iSBC 534		Not applicable.
iSBC 544		If your board does not have a switch SW1, then refer to Table 6-3.
	SW1, 1-4 ON	Selects Dual-Port RAM address. Also refer to Table 6-4.
	SW1, 5 ON SW1, 6 OFF	Selects Dual-Port RAM size of 16K.
	SW1, 7 ON	Selects 2732A EPROMS.
	SW1, 8 OFF	Configures board for slave mode.

DIP HEADER CONFIGURATIONS FOR THE RS232C PROTOCOL

Table 6-8 lists the DIP-header configurations you need to supply to implement the RS232C serial protocol. This configuration process involves either soldering wires on a solder style header or inserting wires into a pin-and-socket style header.

Table 6-8. DIP Header Configurations for the RS232C Protocol

Intel Board	DIP Header Jumpers	Description/Function
iSBX 351	3-13 4-14 7-8 5-6 11-12 9-10	Board RxD to Terminal TxD. Board TxD to Terminal RxD. Board DSR to Board DTR. Board RTS to Board CTS. Terminal RTS to Terminal CTS. Terminal DSR to Terminal DTR.
iSBC 534	4-5 6-7 8-10 9-11 12-13 14-15	Board DSR to Board DTR. Board RTS to Board CTS. Board RxD to Terminal TxD. Board TxD to Terminal RxD. Terminal RTS to Terminal CTS. Terminal DSR to Terminal DTR.
iSBC 544	2-3 4-5 6-12 7-13 14-15 16-17	Board DSR to Board DTR. Board RTS to Board CTS. Board RxD to Terminal TxD. Board TxD to Terminal RxD. Terminal RTS to Terminal CTS. Terminal DSR to Terminal DTR.

Notes: Signal Names:

TxD:	Transmit Data	RxD:	Receive Data
DTR:	Data Terminal Ready	DSR:	Data Set Ready
RTS:	Request To Send	CTS:	Clear To Send

MISCELLANEOUS JUMPERS

Table 6-9 lists jumpering information not covered in the previous sections. The list of jumpers change different functional areas. Perform the changes to use default values established by Intel.

FIRST-TIME INSTALLATION OF THE OPERATING SYSTEM

This section details the steps that you must take to install the iRMX 86 Operating System. Before you begin, make certain that you know which Intel microprocessor runs your system; the installation procedure is changes slightly depending on the processor in your system. If at any time you see an error massage, stop the installation procedure and correct the problem.

CAUTION

> THE INSTALLATION PROCEDURE DESCRIBED HERE AUTOMATICALLY RE-FORMATS THE WINCHESTER DISK DRIVE. Therefore, back up any files you wish to save before you begin the installation procedure.

STEP 1: RUNNING THE SYSTEM CONFIDENCE TEST (SCT)

If you are using your own custom iRMX 86 development system, you should now turn on the power to your system. If you have built an iAPX 86-based development system, you will see a series of asterisks. If you have built an iAPX 286-based development system, you will see no display. In either case you should then type in an uppercase "U". Typing in an uppercase "U" initializes the Monitor and causes it to sign-on. Once you see the monitor prompt--a period (".")--go to Step 2.

If you are using a System 300 product, turn on the power for your system. In about 5 seconds you will see a display. If you have a System 86/300 Series microcomputer, the display will be a series of asterisks. If you have a System 286/300 Series microcomputer, the display will be a single asterisk. In either case, type in an uppercase "U". This will cause the System Confidence Test (SCT) to execute. The SCT is only provided on System 300 Series Microcomputers.

After you type in an uppercase "U", you will see on the CRT display status reports from the SCT. For specific information on the meaning of the reports consult the SYSTEM 86/300 SERIES DIAGNOSTIC MAINTENANCE MANUAL or the SYSTEM 286/300 SERIES DIAGNOSTIC MAINTENANCE MANUAL.

Shortly after beginning the display on systems based on the iAPX 86 microprocessor, the SCT on System 86/300 products requests you to enter an uppercase "I" in response to the "PIC" test. At this point you have three options: 1) do nothing; 2) type in an uppercase "I"; or 3) press the front panel interrupt button. On a system with an un-formatted Winchester disk, all three actions have the same result--you exit the SCT and enter the monitor.

The system should respond by displaying:

```
*BREAK* at <xxxx:yyyy>
.
```

The period (".") is the monitor prompt, and <xxxx:yyyy> is the address where the entry into the monitor occurred. At this point you are ready to go on to the next step.

Once the execution of the SCT on a System 286/300 product begins, the terminal requests you to enter a response to the 8274 MPSC test. After displaying the test prompt, the SCT waits for your response. Respond with a period (".") if you want to enter the monitor at the end of the execution of the SCT rather than boot the iRMX 86 system. If you do not enter a period within six seconds, the test times out and responds with the message "Chb Interrupt Timeout". Normally in the enhanced mode, you may enter any character at the 8274 MPSC prompt and the bootstrap loader boots the operating system after the SCT executes. However, when installing the Operating System, you need to return to the monitor after running the SCT. Entering a period at the 8274 MPSC test prompt signals the SCT to do this. The system should respond by displaying:

```
    *BREAK* at <xxxx:yyyy>
    .
```

The period (".") is the monitor prompt, and <xxxx:yyyy> is the address where the entry into the monitor occurred. At this point you are ready to go on to the next step.

NOTE

> If you are installing the Release 6 version of the Operating System on hardware that is already running the Release 5 version of the iRMX 86 Operating System, do not let the SCT run to completion. When the SCT requests that you input the uppercase "I", you must press the front panel interrupt button. If you do not press the interrupt button, you won't enter the monitor.

STEP 2: INSTALLING THE FIRST DISKETTE

Place the diskette with the label INSTALLATION DISKETTE in the flexible diskette drive. If you have a system based on the iAPX 86 microprocessor, place the diskette with the label iRMX 86 INSTALLATION DISKETTE FOR iAPX 86-BASED SYSTEMS into the flexible diskette drive. If you have a system based on the iAPX 286 microprocessor, place the diskette with the label iRMX 86 INSTALLATION DISKETTE FOR iAPX 286-BASED SYSTEMS into the flexible diskette drive.

Enter the following monitor command depending on the hardware in your system:

HARDWARE IN YOUR SYSTEM	ENTER
iSBC 218A Board mounted on any iSBC 215 that controls either a 5 1/4 or 8-Inch Flexible Disk Drive	.b :wf0:
iSBX 208 Controller	.b :af0:

If you have any SYSTEM 300 Series Microcomputer, use the command b :wf0:.

The Monitor command boots the file "/system/rmx86" from the INSTALLATION DISKETTE so that the INSTALLATION DISKETTE is the system device. The INSTALLATION DISKETTE only contains those Human Interface commands required to initiate the installation process.

Upon completion of the bootstrap load process, the terminal displays the following message:

```
iRMX 86 HI CLI, Vx.y: USER=65535
Copyright <years> Intel Corporation
-
```

Next, the system prompts you for the correct date and time. You may enter the date in any one of the following three formats:

```
month/date/year (11/29/1984)
date month year (29 NOV 1984)
date month year (29 NOVEMBER 1984)
```

After you have entered the date, the system echoes the information and prompts you for the time. Enter the time in the format HOURS:MINUTES:SECONDS. You may omit the minutes and seconds fields if you desire; the system sets them to zero. When you have completed entering the time, the system responds by echoing the entered time. After the date and time are entered and echoed, the system displays the line:

```
END SUBMIT :PROG:R?LOGON
```

STEP 3: BECOME THE SYSTEM MANAGER

To continue the installation process, you need to gain access to the system manager privileges. Enter the command SUPER to gain the power of system manager. In response to the password prompt, enter in a carriage return. The system responds with the prompt "SUPER-". You now have access to the system manager privileges and you may continue with the installation process.

STEP 4: INSTALLING iRMX™ 86 FILES ON A WINCHESTER DISK

During this phase of the installation process you will be formatting your Winchester drive and copying the iRMX 86 files from the Installation Diskette to your formatted Winchester disk. Only those files necessary for booting the operating system from your Winchester disk are copied at this time. To install these essential files, enter the command:

```
SUBMIT /INSTAL(device name, interleave, files)
```

"Device name" is the physical name of the device that boots the operating system after installation (this device is also known as the system device). Refer to Table 10-1 for the correct device names. Do not use the generic name for the device. You must use the name corresponding to the actual device. For example, if you have an 8-inch 30MB Priam Winchester in your system, you must use the device name "iw0" or "iwl".

"Interleave" is "4" for 5 1/4-inch Winchesters and "3" for 8-inch Winchesters.

"Files" is the number of files you want to be able to create on your Winchester disk. A number between 3000 and 6000 should be selected. This number is dependant on your application. Generally, if you have a 10MB or 15MB Winchester drive, create 3000 files. If you have a larger Winchester drive, create 4000 files. If you have purchased the source code from Intel for the iRMX 86 Operating System, you must specify exactly 5000 files.

Table 10-1. Start-Up System Device Names of Winchester Drives

Device	Device Name
CMI 5 1/4" 10 MB Winchester (formatted)	cm0, cml
CMI 5 1/4" 15 MB Winchester (formatted)	cmb0, cmbl
Quantum 5 1/4" 40MB Winchester (un-formatted)	qma0, qmal
8" 30MB Priam Winchester (formatted)	iw0, iwl
8" 70MB Priam Winichester (un-formatted)	iwb0, iwbl

STEP 5: BOOTING THE OPERATING SYSTEM FROM A WINCHESTER DISK

After the system has executed the submit command described in Step 4, the Winchester disk contains enough iRMX files to boot the Operating System and to use selected Human Interface commands. However, before you can boot the Operating System from the Winchester disk, you must remove the INSTALLATION DISKETTE and reset the system. Reset the system by pressing the front panel RESET button or by whatever means you have designed into the system.

If you have a System 300 Series Microcomputer, you will see the display described in Step 1 in about 6 seconds. This time do not type an uppercase "U" in response to the asterisk(s). After about 12 seconds, the SCT will time out and the Terse mode of the SCT will execute. Allow the SCT to run (this verifies that all your hardware is operating correctly). After the SCT successfully executes, the bootstrap loader automatically boots the iRMX 86 Operating System that you copied to the Winchester disk in Step 4.

If you are using a custom built iRMX 86 development system, you must type in an uppercase "U" in response to the Monitor's display described in Step 1. You will then see the Monitor's prompt ("."). At this point, type "b" to boot the iRMX 86 Operating System that you copied to the Winchester disk in Step 4.

Once the operating system loads, the system again prompts you for the date and time. Enter the correct date and time according to the instructions given in Step 2.

To complete the Operating System installation, you need privileges of the system manager. Enter the SUPER command; the system prompts you for the correct password. In response to the password prompt, enter a carriage return. The system responds with the prompt "super-". Now you have the privileges of the system manager and may complete the iRMX 86 Operating System installation.

STEP 6: INSTALLING THE REMAINING iRMX™ 86 FILES

Now that you have successfully booted the operating system from the Winchester disk, install the iRMX 86 files from the remaining seven iRMX 86 Operating System diskettes.

CAUTION

If you are using a system equipped with 5 1/4-inch flexible diskette drives, YOU MUST REMEMBER TO PERFORM THE FOLLOWING STEPS FOR EACH DISKETTE USED:

(1) Insert the diskette into the diskette drive.
(2) Attach the device using the ATTACHDEVICE command.
(3) Use the diskette.
(4) Detach the device using the DETACHDEVICE command.
(5) Remove the diskette from the diskette drive.

You must detach and re-attach the 5 1/4-inch flexible diskette drives with each diskette installation because the I/O System cannot detect the "door open" condition and does not know when the I/O System buffers contain invalid data from a previous diskette. You do not need to detach and re-attach the device for each 8-inch diskette installation.

A. Before installing the remaining iRMX files, you must "attach" the flexible diskette drive to the system by using the following command:

```
super- ATTACHDEVICE device name AS :logical name:
```

The physical name of the flexible diskette drive is "device name" and ":logical name:" is the name the system uses to address the flexible diskette drive. Attach a 5 1/4-inch diskette drive in a System 300 Series Microcomputer using the command:

```
super- ATTACHDEVICE wmfdx0 AS :fd0:
```

Attach an 8-inch flexible diskette drive in a System 300 Series Microcomputer using the command:

```
super- ATTACHDEVICE wfd0 AS :fd0:
```

B. To copy the remaining iRMX 86 files from diskettes number 1 through 7 (listed in Table 1-2), insert the next diskette into the flexible diskette drive and enter the following command:

```
super- SUBMIT :logical name:INSTAL(:logical name:)
```

Both logical names in the preceding SUBMIT command are the same: the system uses these names to address the device as specified in the preceding ATTACHDEVICE command (typically :fd0:). Each diskette has a file named INSTAL.CSD which, when executed, will copy the contents of that diskette into the correct directory on the Winchester disk.

C. Remember, if you have 5 1/4-inch flexible disk drives, you must detach the device using the DETACHDEVICE command. The DETACHDEVICE command has the syntax "DETACHDEVICE :logical name:".

Repeat steps A through C for all of the remaining Release Diskettes.

As the SUBMIT command installs the Operating System files, a series of messages appear. If the system encounters an error during the process, it displays an error message but does not stop: the system continues executing the SUBMIT command until it reaches the end of the process. Watch these messages and be alert for error messages. When the system displays an error message, stop the system and correct the fault.

STEP 7: INSTALLING THE LANGUAGE UTILITIES

The next step is to install the language utilities. If you do not have a System 300 Series Microcomputer from Intel, you must purchase the language products in addition to the iRMX 86 Operating System.

At this point you must still have system manager privileges. (You must still be in SUPER.)

Detach the flexible disk drive before installing the Language Utilities. To do this, use the DETACHDEVICE command.

Before beginning the installation of the language utilities, check that you have the proper diskettes. You should have the following diskettes:

- iRMX 860 ASM86 AND NUMERICS LIBRARIES
- iRMX 860 UTILITIES PACKAGE
- iRMX 863 PL/M-86

The order that you install the diskettes is important. You must install the diskettes in the order they are presented in the list above.

First, place the diskette with the label iRMX 860 ASM86 AND NUMERICS LIBRARIES into the flexible disk drive. Enter in the following SUBMIT command to install the diskette:

```
    SUBMIT /CONFIG/CMD/INSTAL860(<devicename>)
```

<devicename> is the physical device name for the flexible disk drive. The device name for 8-inch diskettes is wfd0 and the device name for 5 1/4-inch diskettes is wmfdx0. Note, the device name is not a logical name so it does not have colons surrounding it.

Second, take out the first diskette (iRMX 860 ASM86 AND NUMERICS LIBRARIES) and place into the flexible disk drive the diskette with the label iRMX 860 UTILITIES PACKAGE. Next, enter in the following SUBMIT command to install the diskette:

```
    SUBMIT /CONFIG/CMD/INSTAL860u(<devicename>)
```

<devicename> is the physical device name for the flexible disk drive.

Third, take the second diskette (iRMX 860 UTILITIES PACKAGE) out of the disk drive and place into the drive the diskette with the label iRMX 863 PL/M-86. Next, enter in the following SUBMIT command to install the last diskette:

```
    SUBMIT /CONFIG/CMD/INSTAL863(<devicename>)
```

<devicename> is the physical device name for the flexible disk drive.

STEP 8: INSTALLING THE UPDATE PACKAGE

The final phase of installing the iRMX 86 Operating System is the installation of the current iRMX 86 Release 6 update package. You must perform this step even if you are installing a new system. Applying the update package is Intel's mechanism for fixing problems identified in the current version of the software. Failure to apply the update results in the installation of an un-fixed version of the iRMX 86 Operating System.

The update package accompanies all shipments of the iRMX 86 Operating System. (The update package is shipped in a separate box.) Each update package contains one or more update diskettes, one or more shrinkwrapped packets of documentation change pages, a customer letter, and an update installation guide. (Occasionally, additional documentation may be supplied in response to special circumstances.)

The Update Diskettes contain all of the fixes (ZAP's) that are to be applied to the iRMX 86 Operating System. The diskettes are labelled:

"RMX86w Rx.y UP z"

where: w is the media type (B, E, or J),

x is the release level of the Operating System,

y is the revision level of the Operating System,

z is the release level of the Update Package.

The update installation guide contains both detailed descriptions of each ZAP and detailed instructions on installing the Update Package.

To install the Update to your system, find the Update Package and follow the instructions in the update installation guide.

REFERRING TO OTHER MANUALS BEFORE RUNNING YOUR SYSTEM

Once you have completed the eight steps listed in the previous section you are ready to use your iRMX 86 Operating System. Refer to the following manuals for additional help:

- For basic information about your system and the manuals in your Release 6 documentation set, refer to the INTRODUCTION TO THE iRMX 86 OPERATING SYSTEM.

- For information about memory partition sizes and further insights into your Start-up System (including information on how to generate a custom Operating System), refer to the iRMX 86 CONFIGURATION GUIDE.

***

4) Bootstrap Loader

Libraries	Includes	Other
bsl.lib	bsl.inc	boot.060
	b204.inc	b204.a86
	b206.inc	b206.a86
	b208.inc	b208.a86
	b215.inc	b215.a86
	b218a.inc	b218a.a86
	b251.inc	b251.a86
	b254.inc	b254.a86
	bsasi.inc	bsasi.a86
	bscsi.inc	bscsi.a86
	bsldev.inc	bcico.obj
	bserr.inc	bsl.a86
	bcico.inc	bserr.a86
		bcsdm.a86
		bsl.csd
		bsl.mp2
		bsl

5) System Debugger

Libraries	Includes	Other
sdb.lib		sdb.030

DISKETTE 4: iRMX 86 HUMAN INTERFACE COMMANDS

1) instal.csd

had.r86	hatach.r86	hback.r86
hcopy.r86	hcrdir.r86	hdate.r86
hdcopy.r86	hdd.r86	hdeb.r86
hdelet.r86	hdir.r86	hdtach.r86
hform.r86	histat.r86	hjobdl.r86
hlocdt.r86	hlock.r86	hlogs.r86
hmem.r86	hpath.r86	hprmt.r86
hrest.r86	hrname.r86	hsbmt.r86
hsuper.r86	htime.r86	hucopy.r86
hvers.r86	hdvfy.r86	hwhoam.r86
hupcpy.csd	hi.030	

DISKETTE 5: iRMX 86 ICU (part 1 of 2), FILES UTILITY AND PATCH UTILITY

1) instal.csd

2) ICU (Part 1 of 2)

   icu86.020
   icu86.86
   rmx86.def

3) Files Utility

   files.041
   files
   files.lnk
   floc.csd
   fs86.def
   fs186.def

4) Patch Utility

   ptch86.023
   ptch86.86
   patch.csd
   patch.cmd
   patch.a86

DISKETTE 6: iRMX 86 ICU (Part 2 of 2), UDI AND CRASH ANALYZER DISKETTE

1) instal.csd

2) ICU

   icu86.020
   icu86.862
   icu86.hlp
   rmx286.def

3) UDI

Libraries	Includes	Other
udi.lib		udi.030

4) Crash Analyzer

Libraries	Includes	Other
sdumpr.lib		scrs86.011 scrs86.86

DISKETTE 7: iRMX 86 Include Files, Interface Libraries and ICU System Definition Files

1) instal.csd

2) All the iRMX 86 Interface Libraries are on this diskette.

```
rpifc.lib     rpifl.lib
ipifc.lib     ipifl.lib
epifc.lib     epifl.lib
lpifc.lib     lpifl.lib
hpifc.lib     hpifl.lib
compac.lib    large.lib     small.lib
```

3) All the iRMX 86 exception code literal files are on this diskette.

```
nexcep.lit    ldwptr.lit
iexcep.lit    ltksel.lit
eexcep.lit    ltkwrd.lit
lexcep.lit
hexcep.lit
uexcep.lit
```

4) All the iRMX 86 System Call External Declaration Include files are on this diskette:

```
hcrccn.ext    hdlccn.ext    hfmtex.ext    hgtchr.ext
hgtcmd.ext    hgticn.ext    hgtipn.ext    hgtocn.ext
hgtopn.ext    hgtpar.ext    hsncmd.ext    hsncor.ext
hsneor.ext    hstpbf.ext    iaatfl.ext    iachac.ext
iaclos.ext    iacrdr.ext    iacrfl.ext    iadlcn.ext
iadlfl.ext    iagtcs.ext    iagtde.ext    iagted.ext
iagtfs.ext    iagtpc.ext    iaopen.ext    iaread.ext
iarnfl.ext    iaseek.ext    iaspec.ext    iasted.ext
iatrun.ext    iawrit.ext    icrioj.ext    icrusr.ext
idlusr.ext    iexioj.ext    igtlds.ext    igtpfx.ext
igttim.ext    igtusr.ext    ihdtdv.ext    iinusr.ext
ilatdv.ext    ildtdv.ext    ipatdv.ext    ipdtdv.ext
isatfl.ext    ischac.ext    isclos.ext    iscrdr.ext
iscrfl.ext    isctcn.ext    isdlcn.ext    isdlfl.ext
isgtcs.ext    isgtfs.ext    islucn.ext    isopen.ext
isrdmv.ext    isrnfl.ext    isseek.ext    isspec.ext
istioj.ext    istpfx.ext    istrun.ext    isttim.ext
istusr.ext    isuncn.ext    iswrmv.ext    iupdat.ext
iwtio.ext     lalioj.ext    laload.ext    lslioj.ext
lsovly.ext    nacctl.ext    nalcmp.ext    ncrcmp.ext
ncrext.ext    ncrjob.ext    ncrmbx.ext    ncrreg.ext
ncrseg.ext    ncrsem.ext    ncrtsk.ext    nctobj.ext
ndlcmp.ext    ndlext.ext    ndljob.ext    ndlmbx.ext
ndlreg.ext    ndlseg.ext    ndlsem.ext    ndltsk.ext
ndsabl.ext    ndsdln.ext    neinit.ext    nenabl.ext
nendln.ext    nenint.ext    nexint.ext    nfrcdl.ext
ngtexh.ext    ngtlev.ext    ngtpat.ext    ngtpri.ext
ngtsiz.ext    ngttok.ext    ngttyp.ext    nincmp.ext
nluobj.ext    noffsp.ext    nrcctl.ext    nrcmes.ext
nrcuni.ext    nrsint.ext    nrstsk.ext    nsgex.ext
```

```
nsgint.ext    nsleep.ext    nsnctl.ext    nsnmes.ext
nsnuni.ext    nstexh.ext    nstint.ext    nstosx.ext
nstpmn.ext    nstpri.ext    nsutsk.ext    nucobj.ext
nwtint.ext    ualloc.ext    uatach.ext    uchac.ext
uchext.ext    uclose.ext    ucreat.ext    udcex.ext
udctim.ext    udelet.ext    udetac.ext    uexit.ext
uflinf.ext    ufree.ext     ugtarg.ext    ugtcs.ext
ugtexh.ext    ugtsid.ext    ugtsiz.ext    ugttim.ext
uopen.ext     uovly.ext     uread.ext     urenam.ext
ursiom.ext    useek.ext     uspecl.ext    uswbf.ext
utrapc.ext    utrpex.ext    utrunc.ext    uwrite.ext
```

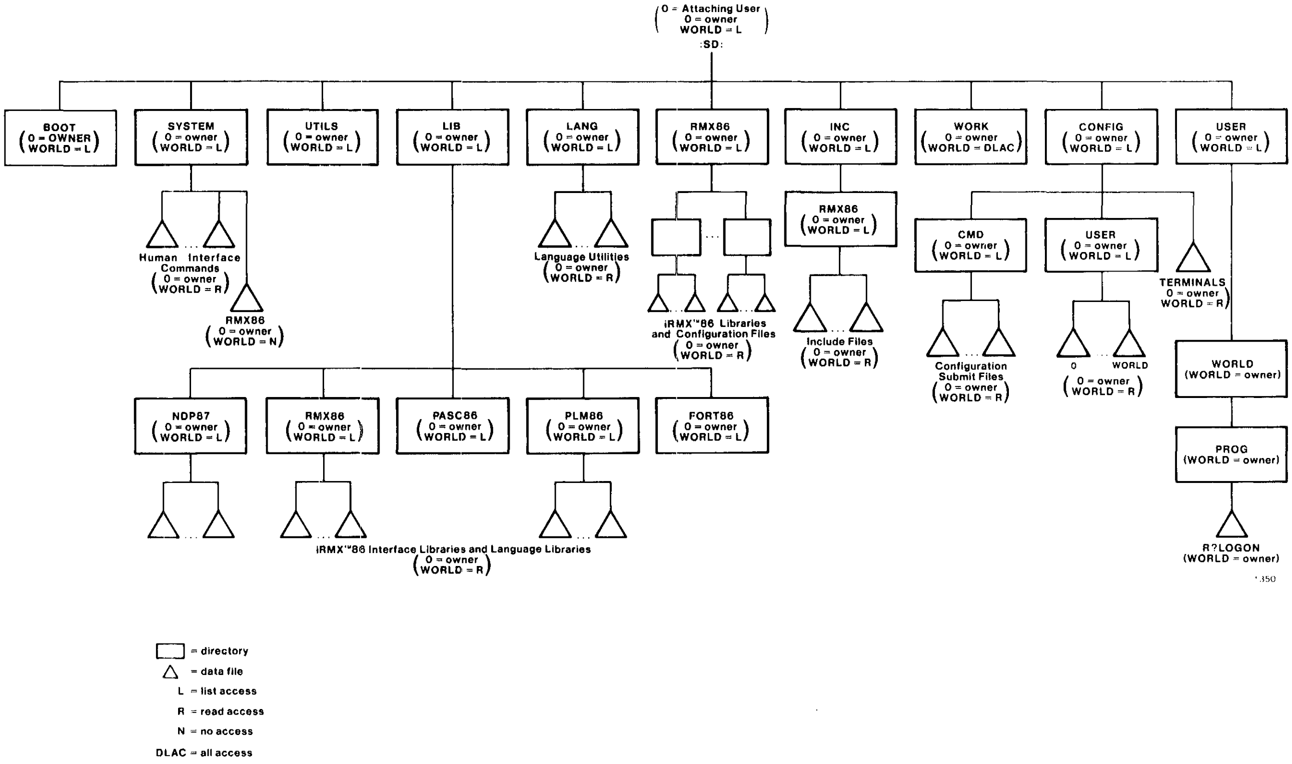
5) Three of the iRMX 86 R6.0 ICU System definition files are on this diskette.

r18603.def
r18651.def
r18848.def

***

APPENDIX E
DIRECTORY STRUCTURE OF THE START-UP SYSTEM

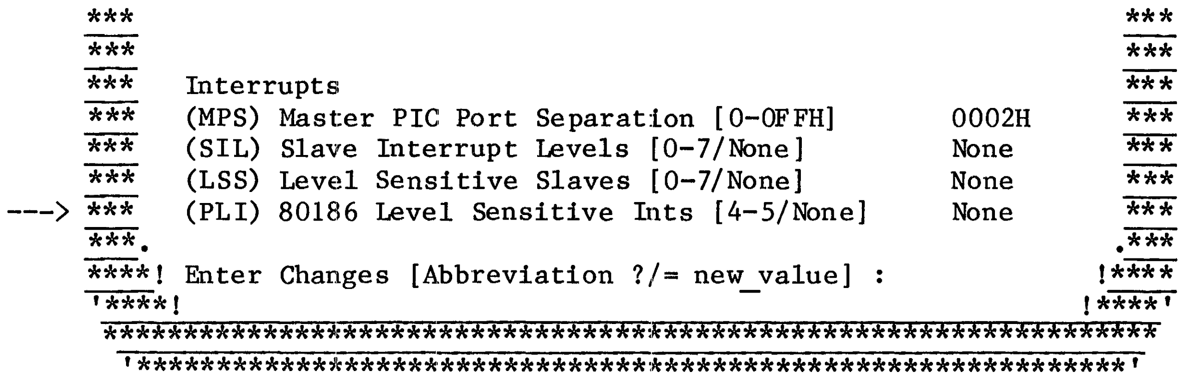
This section shows you the Start-Up system directory structure that exists after you have successfully installed the operating system. This Start-Up directory is in Figure E-1.



(0 = Attaching User 0 = owner WORLD = L)
:SD:
BOOT (0 = OWNER WORLD = L)
SYSTEM (0 = owner WORLD = L)
UTILS (0 = owner WORLD = L)
LIB (0 = owner WORLD = L)
LANG (0 = owner WORLD = L)
RMX86 (0 = owner WORLD = L)
INC (0 = owner WORLD = L)
WORK (0 = owner WORLD = DLAC)
CONFIG (0 = owner WORLD = L)
USER (0 = owner WORLD = L)
Human Interface Commands (0 = owner WORLD = R)
RMX86 (0 = owner WORLD = N)
Language Utilities (0 = owner WORLD = R)
iRMX™86 Libraries and Configuration Files (0 = owner WORLD = R)
RMX86 (0 = owner WORLD = L)
Include Files (0 = owner WORLD = R)
CMD (0 = owner WORLD = L)
USER (0 = owner WORLD = L)
TERMINALS 0 = owner WORLD = R
Configuration Submit Files (0 = owner WORLD = R)
0
WORLD
(0 = owner WORLD = R)
WORLD (WORLD = owner)
PROG (WORLD = owner)
R?LOGON (WORLD = owner)
NDP87 (0 = owner WORLD = L)
RMX86 (0 = owner WORLD = L)
PASC86 (0 = owner WORLD = L)
PLM86 (0 = owner WORLD = L)
FORT86 (0 = owner WORLD = L)
iRMX™86 Interface Libraries and Language Libraries (0 = owner WORLD = R)
*350
□ = directory
△ = data file
L = list access
R = read access
N = no access
DLAC = all access

Figure E-1. Start-Up System Directory Structure

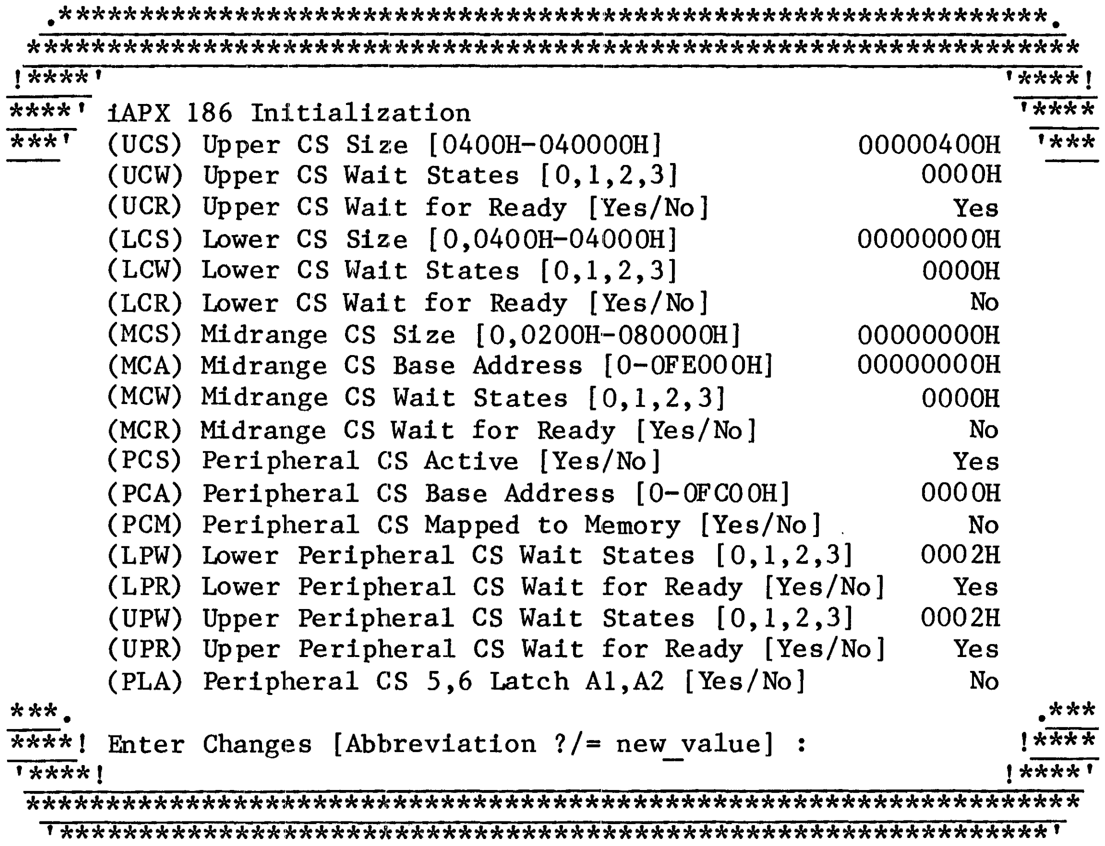
***



```
     ***                                                                ***
     ***                                                                ***
     ***   Interrupts                                                   ***
     ***   (MPS) Master PIC Port Separation [0-0FFH]          0002H     ***
     ***   (SIL) Slave Interrupt Levels [0-7/None]            None      ***
     ***   (LSS) Level Sensitive Slaves [0-7/None]            None      ***
---> ***   (PLI) 80186 Level Sensitive Ints [4-5/None]        None      ***
     ***.                                                              .***
     ****! Enter Changes [Abbreviation ?/= new_value] :                !****
     '****!                                                            !****'
       *****************************************************************
        '***************************************************************'
```

iAPX 186 INITIALIZATION SCREEN

This screen allows you to configure the iAPX 186 or iAPX 188 logic that provides programmable chip-select generation for memories and peripherals. Refer to the iAPX 186 HIGH INTEGRATION 16-BIT MICROPROCESSOR Data Sheet for information about these parameters.



```
    .***************************************************************.
  *******************************************************************
 !****'                                                         '****!
 ****' iAPX 186 Initialization                                   '****
 ***'  (UCS) Upper CS Size [0400H-040000H]          00000400H     '***
       (UCW) Upper CS Wait States [0,1,2,3]             0000H
       (UCR) Upper CS Wait for Ready [Yes/No]             Yes
       (LCS) Lower CS Size [0,0400H-04000H]         00000000H
       (LCW) Lower CS Wait States [0,1,2,3]             0000H
       (LCR) Lower CS Wait for Ready [Yes/No]              No
       (MCS) Midrange CS Size [0,0200H-080000H]     00000000H
       (MCA) Midrange CS Base Address [0-0FE000H]   00000000H
       (MCW) Midrange CS Wait States [0,1,2,3]          0000H
       (MCR) Midrange CS Wait for Ready [Yes/No]           No
       (PCS) Peripheral CS Active [Yes/No]                Yes
       (PCA) Peripheral CS Base Address [0-0FC00H]      0000H
       (PCM) Peripheral CS Mapped to Memory [Yes/No]       No
       (LPW) Lower Peripheral CS Wait States [0,1,2,3]  0002H
       (LPR) Lower Peripheral CS Wait for Ready [Yes/No]  Yes
       (UPW) Upper Peripheral CS Wait States [0,1,2,3]  0002H
       (UPR) Upper Peripheral CS Wait for Ready [Yes/No]  Yes
       (PLA) Peripheral CS 5,6 Latch A1,A2 [Yes/No]        No
 ***.                                                             .***
 ****! Enter Changes [Abbreviation ?/= new_value] :              !****
 '****!                                                          !****'
  *******************************************************************
   '***************************************************************'
```

```
************************************************************************
*       (UCS) Upper CS Size [0400H-040000H]           00000400H      *
************************************************************************
```

You must specify the size of the upper memory chip select line. The value you specify must be 1K (400H), 2K (800H), 4K (1000H), 8K (2000H), 16K (4000H), 32K (8000H), 64K (10000H), 128K (20000H), or 256K (40000H). If you are using the iSBC 186/03 or the iSBC 186/51 processor board, it is recommended that you use the default value. If you are using the iSBC 188/48 processor board, change the default value to 1000OH.

The upper limit defined by this chip select line is always FFFFFH. The lower limit is ascertained by the ICU as the upper limit less the value specified for this parameter line.

```
************************************************************************
*       (UCW) Upper CS Wait States [0,1,2,3]                0000H      *
************************************************************************
```

You must specify the number of wait states for all accesses to the upper memory chip select line. The value you select can be from zero to three. If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value.

```
************************************************************************
*       (UCR) .Upper CS Wait for Ready [Yes/No]               Yes      *
************************************************************************
```

You must select whether or not the iAPX 186 should ignore external READY for the upper memory chip select line. If you specify "Yes", the iAPX 186 will wait for the number of wait states specified or will wait for an external READY condition. If you specify "No", the iAPX 186 will wait for the number of wait states specified but will not wait for an external READY condition. If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value.

```
************************************************************************
*       (LCS) Lower CS Size [0,0400H-040000]          00000000H      *
************************************************************************
```

In response to the "Lower CS Size" parameter line you must specify a value of zero or the size of the lower memory chip select line. The value of zero indicates that you do not intend to program the lower memory chip select line. Any non-zero value you specify must be 1K (400H), 2K (800H), 4K (1000H), 8K (2000H), 16K (4000H), 32K (8000H), 64K (10000H), 128K (20000H), or 256K (40000H). If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value. The lower limit defined by this chip select line is always 00000H. The upper limit is ascertained by the ICU as the lower limit plus the value specified for this parameter line.

```
************************************************************************
*        (LCW) Lower CS Wait States [0,1,2,3]               0000H      *
************************************************************************
```

If you specified a non-zero value for the "Lower CS Size" parameter line, you must specify the number of wait states for all accesses to the lower memory chip select line. The value you select can be from zero to three. If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value.

```
************************************************************************
*        (LCR) Lower CS Wait for Ready [Yes/No]                No      *
************************************************************************
```

If you specified a non-zero value for the "Lower CS Size" parameter line, you must select whether or not the iAPX 186 should ignore external READY for the lower memory chip select line. If you specify "Yes", the iAPX 186 will wait for the number of wait states specified or will wait for an external READY condition. If you specify "No", the iAPX 186 will wait for the number of wait states specified but will not wait for an external READY condition. If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value.

```
************************************************************************
*        (MCS) Midrange CS Size [0,02000H-080000H]      00000000H      *
************************************************************************
```

In response to the "Midrange CS Size" parameter line you must specify a value of zero or the size of the midrange memory chip select line. The value of zero indicates that you do not intend on programming the midrange memory chip select line. Any non-zero value you specify must be 8K (2000H), 16K (4000H), 32K (8000H), 64K (10000H), 128K (20000H), 256K (40000H), or 512K (80000H). If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value.

The iAPX 186 provides four midrange memory chip select lines. Your response to this parameter sets the total size of the memory block defined by the four midrange select lines. The size of any one midrange memory chip select line is one-fourth of the total. The lower limit defined by this chip select line is defined by the "Midrange Chip Select Base Address". The upper limit is ascertained by the ICU as the lower limit plus the value specified for this parameter line.

```
***********************************************************************
*       (MCA) Midrange CS Base Address [0-0FE000H]     00000000H      *
***********************************************************************
```

If you specify a non-zero value for the "Midrange CS Size" parameter line, you must specify the base address of the midrange memory chip select lines. Otherwise, specify a value of zero. If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value.

You must set the base address at any integer multiple of the size of the total memory block selected. For example, if you specified a total block size of 32K for the previous parameter (MCS), you must select a base address of 10000H or 18000H but not 14000H.

If you specify MCS=080000H for the previous parameter line, you must also specify the base address to be 00000H and the "Lower CS Size" parameter to be zero.

```
***********************************************************************
*       (MCW) Midrange CS Wait States [0,1,2,3]            0000H      *
***********************************************************************
```

If you specified a non-zero value for the "Midrange CS Size" parameter line, you must specify the number of wait states for all accesses to the midrange memory chip select lines. Otherwise, specify a value of zero. The value you select can be from zero to three. If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value.

```
***********************************************************************
*       (MCR) Midrange CS Wait for Ready [Yes/No]             No      *
***********************************************************************
```

If you specified a non-zero value for the "Midrange CS Size" parameter line, you must select whether or not the iAPX 186 should ignore external READY for the midrange memory chip select lines. Otherwise, specify a value of zero. If you specify "Yes", the iAPX 186 will wait for the number of wait states specified or will wait for an external READY condition. If you specify "No", the iAPX 186 will wait for the number of wait states specified but will not wait for an external READY condition. If you are using the iSBC 186/03, the iSBC 186/51, or the iSBC 188/48 processor board, it is recommended that you use the default value.

The module with the highest address. Since the Root Job module is always the last module the second stage of the ICU locates, the information we need to complete this table must come from the ROOT.MP2 file. The contents from a sample ROOT.MP2 file is shown in Figure 15-2.

```
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(029FF0H),DATA(02A4B0H)))
  INITCODE(029FF0H) OC(NOCM,NOSB) PC(NLOI,PL,NOXM,NOSB)  OOD
      •
      •
      •
MEMORY MAP OF MODULE RBEGIN

MODULE START ADDRESS PARAGRAPH = 29FFH  OFFSET = 0000H
SEGMENT MAP

START      STOP      LENGTH ALIGN NAME             CLASS

-->29FF0H   2A2F3H     0304H   W   CODE             CODE
 2A2F4H   2A2FFH     0012H   W  SAB_DESCRIPTOR  CODE
                             -S
-->2A300H   2A3C5H     00C6H   W   U_J_DESCRIPTOR  CODE
                             -S
-->2A4B0H   2A4C1H     0012H   W   DATA             DATA
 2A4C2H   2A5EDH     012CH   W  INIT_STACK       STACK
 2A5EEH   2A5EEH     0000H   W  STACK            STACK
 2A5F0H   2A5F0H     0000H   G  ??SEG
 2A5F0H   2A5F0H     0000H   W  MEMORY           MEMORY


GROUP MAP

ADDRESS  GROUP OR SEGMENT NAME
2A4B0H   DGROUP
          DATA
29FF0H   CGROUP
          CODE
          SAB_DESCRIPTORS
          U_J_DESCRIPTORS
```

Figure 15-2. ROOT.MP2 File

The lines marked with arrows in Figure 15-2 contain the sample information we need to complete the table. Since the ICU has organized the modules in the order shown in Figure 15-3, we can also estimate the other needed stop addresses. (Note that the Root Job's data and stack segments should be treated as one block of RAM.)

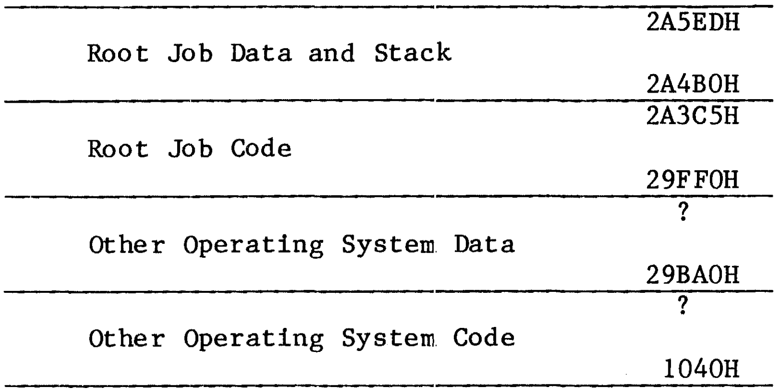


Figure 15-3. A Sample RAM-Based System

The following start addresses summarize this sample information.

System Module	Code Locations	Data Locations
IOS	001040H- ?	029F30H- ?
HI	011620H- ?	029EB0H- ?
NUCLEUS	0177C0H- ?	029FC0H- ?
SDB	01D7A0H- ?	029BA0H- ?
EIOS	022740H- ?	029FE0H- ?
LOADER	025AA0H- ?	029FA0H- ?
UDI	028030H-029B9FH	029F80H-029FEFH
ROOT	029FF0H-02A3C5H	02A4B0H-02A4C1H

Having determined the basic size requirements of the system's code and data segments, we can approximate the RAM and ROM requirements of this sample ROM-based system. Figure 15-4 shows how we can configure our sample ROM-based system.

NOTE

All data segments must be in RAM. All RAM and ROM code must start on a 16 byte boundaries.

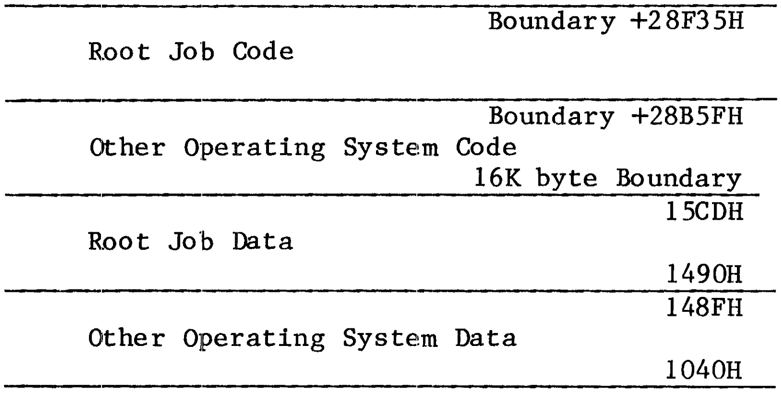
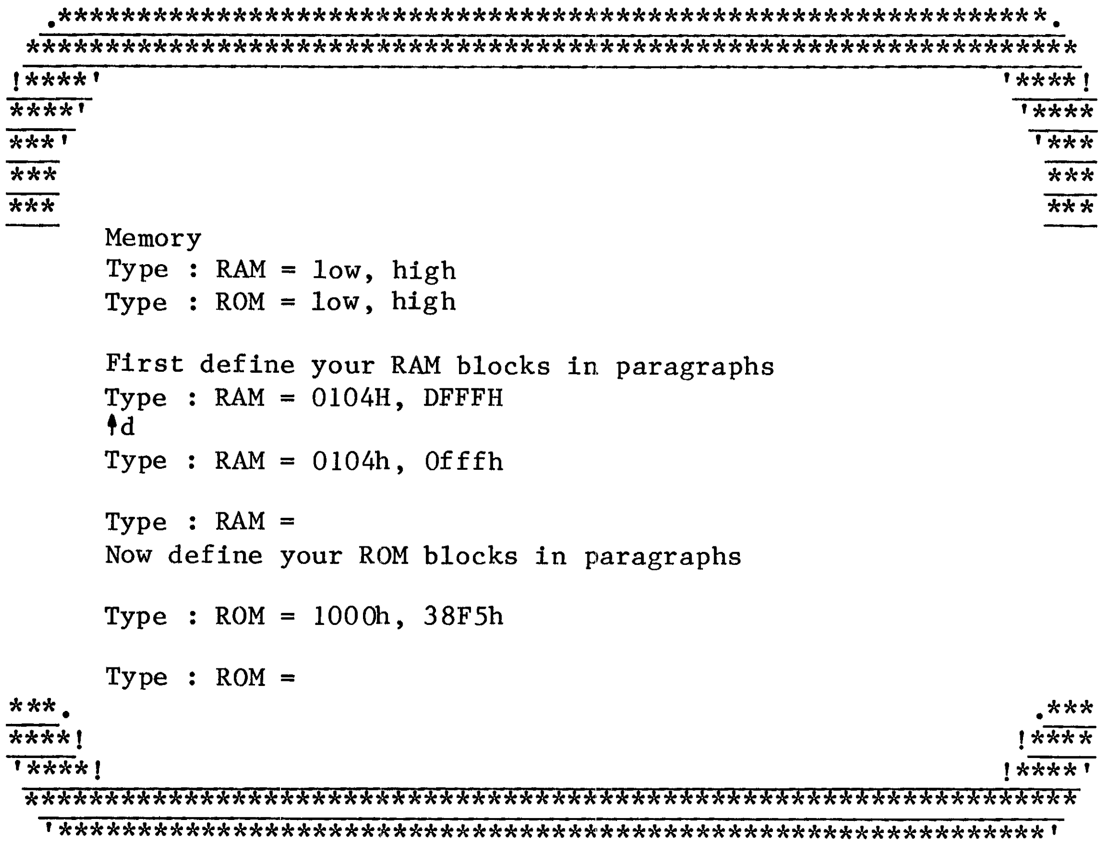


Figure 15-4. A Sample ROM-Based System

Before the system code is burned into ROM, it is recommended that you test your ROM-based system in RAM. To do this, invoke the ICU and respond to the "ROM" prompt with a RAM address in the "Memory" screen. The following screen shows the changes in our sample configuration.



```
Memory
Type : RAM = low, high
Type : ROM = low, high

First define your RAM blocks in paragraphs
Type : RAM = 0104H, DFFFH
↑d
Type : RAM = 0104h, 0fffh

Type : RAM =
Now define your ROM blocks in paragraphs

Type : ROM = 1000h, 38F5h

Type : ROM =
```

ROM CODE PARAMETERS

Running the second stage of the ICU reveals that our sample ROM-based system would have the following RAM addresses:

System Module	Code Locations	Data Locations
IOS	010000H- ?	0013D0H- ?
HI	0205E0H- ?	001350H- ?
NUCLEUS	026780H- ?	001460H- ?
SDB	02C760H- ?	001040H- ?
EIOS	031700H- ?	001480H- ?
LOADER	034A60H- ?	001440H- ?
UDI	036FF0H- ?	001420H- ?
ROOT	038B60H-038F35H	001490H-0015CDH

The last steps you need to take to create a ROM-based system include:

- Use LIB86 to put all generated system modules into the system library for boot loading or down-loading.
- Load and test your ROM-based system in RAM.
- Invoke the ICU to give the "Memory" screen actual ROM addresses.
- Generate your configuration files to link and locate your new system.
- Record the Start and Stop addresses of each module to be burned into ROM. This information is found in the memory maps LOC86 generates for each module.
- Burn your code into ROM.

Appendix C illustrates how to burn your Nucleus code into ROM. Refer to this appendix for more information.

***

iRMX™ 86 RELEASE 6
DOCUMENTATION CHANGE PACKAGE:
UPDATE 3
147540-001

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, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

2. Does the 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 ______________________________

CITY ______________ STATE ______________ ZIP CODE ______________
(COUNTRY)

Please check here if you require a written reply. □

WE'D LIKE YOUR COMMENTS . . .

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

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

OMO Technical Publications