iRMX 86™ SYSTEM DEBUG MONITOR REFERENCE MANUAL

Order Number: 143908-001

REV.	REVISION HISTORY	PRINT DATE
-001	**Original Issue**	12/81

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

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

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

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

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

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

BXP	Insite	iSBC	Multibus
CREDIT	Intel	iSBX	Multimodule
i	int$_e$l	Library Manager	Plug-A-Bubble
ICE	Intelevision	MCS	PROMPT
iCS	Intellec	Megachassis	RMX/80
i^{2}m	iOSP	Micromainframe	System 2000
iMMX	iRMX	Micromap	UPI

PREFACE

This manual describes the iRMX 86 System Debug Monitor (also referred to as the SDB). The SDB is an extension of the iSBC 957B Monitor that allows you to interactively examine your iRMX 86 System in order to find and correct errors. This manual contains introductory and overview material as well as detailed descriptions of all SDB commands.

READER LEVEL

This manual is intended primarily for system programmers who are familiar with both the iRMX 86 Operating System and with the concepts and terminology in the USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE. However, application programmmers will find some of the SDB features helpful in debugging application tasks.

CONVENTIONS

Throughout this manual, the iRMX 86 System Debug Monitor is called the SDB and the iSBC 957B iAPX 86, 88 Interface and Execution Package is called the iSBC 957B Monitor.

Chapter 4 of this manual, which contains detailed descriptions of the commands, uses "CS:IP" to mean "code segment:instruction pointer." This chapter also contains several examples of SDB commands entered at the terminal. In these examples, user input is underscored to distinguish it from SDB output. Carriage returns are not shown after the user input but they are required for the SDB to execute the command.

Unless otherwise noted, all numbers the SDB displays are hexadecimal. All bits are numbered from right to left with the bit on the far right being zero.

RELATED PUBLICATIONS

The following manuals provide additional background and reference information.

Manual	Number
Guide to Writing Device Drivers for the iRMX™ 86 and iRMX™ 88 I/O Systems	142926
iRMX™ 86 Nucleus Reference Manual	9803122
iRMX™ 86 System Programmers Reference Manual	142721
iRMX™ 86 Basic I/O System Reference Manual	9803123
iRMX™ 86 Disk Verification Utility Reference Manual	144133
User's Guide for the iSBC™ 957B iAPX 86, 88 Interface and Execution Package	143979
iRMX™ 86 Configuration Guide	9803124

CONTENTS

FIGURES

CHAPTER 1. ORGANIZATION

This manual is divided into four chapters. Some of the chapters contain introductory or overview material which you might not need to read if you are already familiar with the iSBC 957B Monitor and the iRMX 86 System Debug Monitor (SDB). Other chapters contain reference material which you will refer to as you debug your system. You can use this chapter to determine which of the other chapters you should read.

The organization of the manual is as follows:

Chapter 1	This chapter describes the organization of the manual. You should read this chapter if you are going through the manual for the first time.
Chapter 2	This chapter describes the features of the iRMX 86 System Debug Monitor (SDB) and its relationship to the iSBC 957B Monitor. You should read this chapter if you are going through the manual for the first time.
Chapter 3	This chapter explains how to install and configure the iRMX 86 System Debug Monitor into your system. It also describes how to invoke and execute the iRMX 86 System Debug Monitor (SDB). You should read this chapter if you are installing and/or configuring the SDB into your system.
Chapter 4	This chapter contains detailed descriptions of the iRMX 86 System Debug Monitor (SDB) commands. The commands are listed in alphabetical order for easy referencing. When you are debugging your system, you should refer to this chapter for specific information about the format and parameters of the command.

CHAPTER 2. INTRODUCTION

The development of almost every system requires debugging. To aid in the development of iRMX 86-based application systems, Intel provides three debugging tools: the iRMX 86 Debugger, the ICE-86 In-Circuit Emulator, and the iSBC 957B Monitor. This manual describes how the SDB extends the capabilities of the 957B Monitor.

ADVANTAGES OF THE iRMX 86 DEBUGGER

The iRMX 86 Debugger is a debugging tool which is sensitive to the data structures that the Nucleus maintains. The iRMX 86 Debugger is especially good to use when you want to :

- manipulate or examine one task while other tasks in the system continue to run
- monitor system activity without interfering in the execution
- examine and interpret the data structures associated with the Nucleus and the Nucleus objects

ADVANTAGES OF THE ICE-86 EMULATOR

ICE-86 provides in-curcuit emulation for 8086 microprocessor-based systems. The ICE-86 emulator is especially good to use when you want to:

- get closer to the hardware level by examining the contents of input pins and input ports
- change the values at output ports
- examine individual components rather than an entire board
- look at the most recent 80 to 150 assembly language instructions executed

ADVANTAGES OF THE iSBC 957B MONITOR AND THE SDB

The iSBC 957B Monitor supports both interactive commands and system I/O routines. It allows you to

- disassemble code
- set execution and memory break points
- display memory

You can extend these capabilities by making the iRMX 86 System Debug Monitor (SDB) part of your operating system. In addition to retaining the features of the iSBC 957B, the SDB

- identifies and interprets iRMX 86 system calls
- displays and interprets iRMX 86 objects

REQUIREMENTS OF THE iRMX 86 SYSTEM DEBUG MONITOR

In order to use the SDB, you must have one of the following combinations of equipment:

- a terminal connected directly to an iAPX 86, 88-based board.
- an Intellec system connected to an iAPX 86, 88-based board.

You must also have the monitor portion of the iSBC 957B iAPX 86, 88 Interface and Execution Package, the iRMX 86 System Debug Monitor, and at least the minimal configuration of the Nucleus. The SDB needs only a small portion of valid Nucleus code so most of the SDB commands will function even if you accidentally write over part of the Nucleus. This allows you to continue debugging even if a large portion of the Nucleus has been overwritten or destroyed. See Chapter 3 for more information on configuring and installing the SDB.

CHAPTER 3. INSTALLATION AND CONFIGURATION

There are three ways to include the SDB in your application system. You can:

- configure the SDB into your system so that every time you bring your system up you also have the SDB
- load the stand-alone SDB module from a Development System
- boot the stand-alone SDB module from an iRMX 86 device

If you intend to configure the SDB into your system, you must first configure the interrupt level that the SDB is to use, assemble the configuration module, and then link and locate the SDB. If you intend to load or bootstrap load the SDB, you do not need to configure the interrupt level, but you must locate the SDB at an absolute address. This chapter describes these processes as well as the steps for loading or bootstrap loading the SDB.

HOW THE SDB IS SUPPLIED

The SDB is available on Release Diskettes in either an ISIS-II format or an iRMX 86 format. If you use the ISIS-II format you will perform the configuration on a Development System. If you use the iRMX 86 format, you will work on your target system.

CONFIGURING THE INTERRUPT LEVEL

If you wish to include the SDB as part of your application system, you can use a built-in interrupt handler that allows you to enter the iSBC 957B Monitor by pressing your interrupt button providing that the interrupt button is connected to the corresponding interrupt level. The interrupt level for the interrupt handler is configurable. The default level is interrupt level 1 which is encoded as 18h. If you want to use a different interrupt level, change "018h" to the correct encoded iRMX 86 interrupt level in the configuration module SDBCNF.A86 (shown in Figure 3-1).

```
name        sdbcnf

data_fpi    segment     public 'DATA'
data_fpi    ends

            assume      ds:data_fpi

            public      level

data_fpi    segment

level       dw          018h            ;change "018h" to the correct
                                        ;encoded interrupt level

data_fpi    ends

            end
```

Figure 3-1. Configuration Module (SDBCNF.A86)

If you decide to use an interrupt level other than the default, be sure that the interrupt button is also connected to this level. See the Nucleus System Call, RQSETINTERRUPT, and the Interrupt Management chapter in the iRMX 86 NUCLEUS REFERENCE MANUAL for more information on interrupt levels.

If you do not plan to include the SDB as part of your system, the configuration of the interrupt level has no effect. This is because the SDB uses the Nucleus system call SET$INTERRUPT to set the interrupt level and if the SDB is not part of your system, there may be no Nucleus. The only way you can interrupt into the monitor, once your system is running, is if your interrupt button is connected to the non-maskable interrupt.

ASSEMBLING, LINKING, AND LOCATING

After you have set the interrupt level to the correct value, invoke the SUBMIT file SDB.CSD as follows:

```
-SUBMIT :device:SDB.CSD (loc_address)
```

where :device: identifies the device containing the SDB.CSD file. If you are using an iRMX 86-based system, you might also have to enter additional path components. This SUBMIT command assembles the configuration module (SDBCNF.A86), links it to the SDB, and locates the entire SDB at the address you specify in "(loc_address)." Even if you do not plan to include the SDB as part of your application system, you must still use this SUBMIT command to locate the SDB. However, do not locate the SDB at an address that will cause it to overlay other parts of the system.

CONFIGURING THE SDB INTO YOUR SYSTEM

You can include the SDB as part of your iRMX 86 application system by going through the configuration and the root job generation processes. (The process of configuring your application system and generating the iRMX 86 root job is throughly explained in the iRMX 86 CONFIGURATION GUIDE.)

Use a %SAB macro to allocate space for the SDB so the Nucleus won't assign this memory as free space. Now, assign the SDB as a first-level job using the locate address of the SDB as the start address of the job in the %JOB macro. An example of the SDB root %JOB macro is as follows:

```
%JOB(0,                     %' object directory size
     0100h, 0100h,          %' pool size (min., max.)
     010h, 010h,            %' max objects and tasks
     0,                     %' max job priority
     0:0, 0,                %' exception handler addr, mode
     0,                     %' job flags
     130,                   %' init task priority,
     loc_address,           %' init task entry address
     0,                     %' init task data segment address
     0:0, 100h,             %' init task stack address, stack size
     0)                     %' init task flags
```

You would place the start address in the "loc_address" of the %JOB macro in the previous example. (See the iRMX 86 CONFIGURATION GUIDE for more information about %JOB and %SAB macros.) When the system starts running, a task in the job created as a result of the %JOB macro initializes the SDB. The task then deletes the entire job that was created as a result of the %JOB macro and all the job's resources are released to the system.

USING THE INTERRUPT

If you configure the SDB into your system, you can interrupt your application system while it is running by pushing the interrupt button. When you press the interrupt button, your program stops at the current address of whatever instruction was being executed. The iAPX 86, 88 Monitor then displays that address on your screen along with a prompt. You can now use the iSBC 957B and SDB commands to debug your program.

LOADING THE SDB FROM A DEVELOPMENT SYSTEM

If you are using a development system, you can load the SDB into memory by entering the iSBC 957B load (L) command as follows:

```
.L :fx:SDB
```

where ":fx:" is the disk identifier that corresponds to the release disk which contains the SDB and "SDB" is the file on the release diskette that

contains the located iRMX 86 System Debug Monitor. You obtained this file as a result of submitting SDB.CSD, as described previously. The load command (L) automatically sets the CS and IP registers to the SDB starting address.

Next, use the iSBC 957B go command (G) to automatically initialize the SDB and return to the iSBC 957B Monitor.

```
.G
```

The system responds with the SDB sign-on message and a prompt.

```
iRMX 86 SYSTEM DEBUG MONITOR, Vx.x
.
```

An alternative to using the load (L) and go (G) commands is to use the load-and-go command (R). You can enter this command as follows:

```
.R :fx:SDB
```

where :fx: is the disk identifier that corresponds to the release disk which contains the SDB. This command functions identically to using the load (L) and go (G) commands separately.

You can now enter any iSBC 957B or SDB commands. If you want to run your application system, you must enter the iSBC 957B go command (G) with the starting address of your system.

NOTE

After you start your iRMX 86-based system running, there is no way to return to the SDB or the iSBC 957B Monitor unless:

- you have set breakpoints
- you use the Human Interface DEBUG command (see "Using the DEBUG Command" in this chapter)
- your interrupt switch is connected to the non-maskable interrupt

BOOTSTRAP LOADING THE SDB FROM AN iRMX 86 DEVICE

If the iRMX 86 Bootstrap Loader resides in ROM on your hardware system, you can use it to load the SDB. In order to do this you must first place the located SDB in a file on an iRMX 86 device. Then you can load the SDB into memory by using the iSBC 957B bootstrap (B) command as follows:

```
.B sub-pathname
```

where "sub-pathname" is the pathname of the file that contains the located SDB. The Bootstrap command automatically initializes the SDB and the system responds with the a sign-on message and a prompt.

```
iRMX 86 SYSTEM DEBUG MONITOR, Vx.x
.
```

You can now enter any iSBC 957B or SDB commands. If you want to run your iRMX 86 system, you must enter the iSBC 957B go command (G) along with the starting address of your system.

NOTE

After you start your iRMX 86-based system running, there is no way to return to the SDB or the iSBC 957B Monitor unless:

- you have set breakpoints
- you use the Human Interface DEBUG command (see "Using the DEBUG Command" in this chapter)
- your interrupt switch is connected to the non-maskable interrupt

USING THE DEBUG COMMAND

If your Operating System includes the Human Interface, you can debug a file instead of accessing the iSBC 957B Monitor directly. You can use the Human Interface command, DEBUG along with the path name of the file you wish to debug. See the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL for more information concerning the DEBUG command.

When you use the DEBUG command along with a path name of a file, the Operating System returns control to the iSBC 957B Monitor by breaking at the start address of the file. At this point, the Monitor is ready to accept iSBC 957B and SDB commands.

RETURNING TO YOUR APPLICATION

When you have finished debugging your application system, you can let it continue executing by using the iSBC 957B go command (G). See the USER'S GUIDE FOR THE iSBC™ 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE for more information concerning the (G) command.

CHAPTER 4. COMMANDS

This chapter contains some general information concerning the SDB commands plus detailed descriptions of the iRMX 86 System Debug Monitor commands, in aphabetical order. There is also a Command Dictionary which lists the commands in functional groups.

VALIDITY OF A TOKEN

The iRMX 86 Operating System maintains tokens in doubly-linked lists. So, whenever you enter a command that requires a token as a parameter, the SDB automatically checks the validity of that token by looking at the token's forward and backward links. It checks tokens that you enter as parameters for the VD, VK, VJ, VO, VR, and VT commands as well as the tokens that are listed in the displays.

If a token's forward link is bad, the SDB generates a forward link error message along with the information that the particular command usually displays. The token you enter as a parameter of the SDB command always appears in the center of the display. The display for the forward link error is as follows:

```
Forward link ERROR:    4111-->4E85    4111<--4E85-->4155    ?FFFF<--4155
```

The arrows represent links. A right pointing arrow represents a forward link. In this example, you can tell the token 4E85 was used as a parameter since it is in the center of the display and is the only token with an arrow in each direction. The forward link for 4E85 points to 4155. However, the back link from 4155 does not point backward to 4E85, but to FFFF. The question mark before FFFF indicates an error.

If a token's backward link is bad, the SDB generates a back-link error message along with the information the particular command usually displays. The token you enter as a parameter of the SDB command always appears in the center of the display. The display for the backward link error is as follows:

```
Back link ERROR:    4111-->410F?    4111<--4E85-->4155    4E85<--4E53
```

The left pointing arrow respresents a backward link. In this example, you can tell the token 4E85 was used as a parameter since it is in the center of the display and is the only token with an arrow in each direction. The backward link for 4E85 points to 4111. But the forward link from 4111 does not point to 4E85. The question mark after 410F indicates an error.

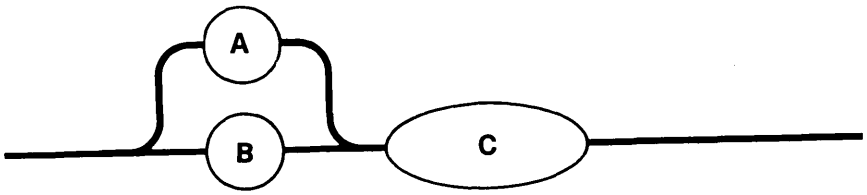
A forward or backward link error means that the iRMX 86 data structures have been damaged or destroyed. The most common reason for this problem is overwriting. You or one of your tasks may have accidentally written over part of the Operating System's data structures and/or code. Another common reason for the problem (if you are using a non-maskable interrupt) could be that you interrupted the Nucleus while it was setting up the links. If either of these things happen, you must re-initialize and reload the SDB (and perhaps your System) before you can use the VD, VJ, VK, VO, VR, or VT commands without getting another link error. See Chapter 3 in this manual for more information on initializing and loading the SDB on your system.

If both links are bad, the SDB considers the token invalid and displays the following message.

```
*** INVALID TOKEN ***
```

PICTORIAL REPRESENTATION OF SYNTAX

This manual uses a schematic device to illustrate the syntax of commands. The schematic consists of what looks like an arerial view of a model railroad setup, with syntactic entities scattered along the track. Imagine that a train enters the system at the far left, drives around as much as it can or wants to (sharp turns and backing up are not allowed), and finally departs at the far right. The command it generates in so doing consists of the syntactic entities that it encounters on its journey. The following pictorial syntax shows two ways (A or B) of reaching "C."



The pictorial syntax of the commands in this chapter does not show spaces as entities. However, the SDB does allow one or more spaces between the command and the parameter. For example, even though the syntax for VR is:

you can enter:

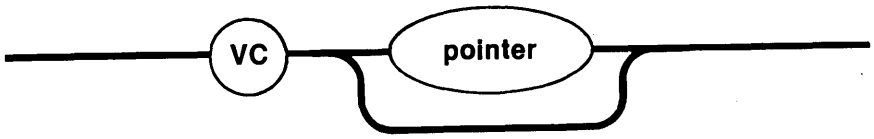
```
.VR xxxx
```

The space between "VR" and "xxxx" does not affect the result of the command.

COMMAND DICTIONARY

VC--Display System Call Information

The VC command checks to see if a CALL instruction is an iRMX 86 system call.



PARAMETER

pointer	The pointer is an optional parameter. It can be any valid iSBC 957B address. If you do not specify a pointer, the System Debug Monitor uses the current CS:IP as the default value. If you do specify an IP value but not a CS value, the SDB uses the current CS as the default base.

DESCRIPTION

If the CALL instruction is an iRMX 86 system call, VC displays information about the CALL instruction as shown in Figure 4-1.

```
S/W int:  xx (subsystem)      entry code xxxx      system call
```

Figure 4-1. Format of VC Output

The fields in Figure 4-1 are as follows:

S/W int: xx (subsystem)	The software interrupt number and the iRMX 86 subsystem that corresponds to that number.
entry code xxxx	The entry code for the system call within the subsystem.
system call	The name of the iRMX 86 system call.

VC (continued)

ERROR MESSAGES

The SDB returns the following error messages for the VC command

Error Message	Description
Syntax Error	You made an error in entering the command.
Not a system CALL	The parameter you specified does not point to a system call.
Not a CALL instruction	The parameter you specified does not point to any kind of call instruction.

EXAMPLES

Suppose you disassembled the following code using the iSBC 957B command, "DX".

```
49A4:006D 50          PUSH        AX
49A4:006E E8AD1E      CALL        A = 1F1E      ;$+7856
49A4:0071 E8DD03      CALL        A = 0451H     ;$+992
49A4:0074 B80000      MOV         AX,0
49A4:0077 50          PUSH        AX
49A4:0078 8D060600    LEA         AX,WORD PRT 006H
49A4:007C 1E          PUSH        DS
49A4:007D 50          PUSH        AX
49A4:007E E8411E      CALL        A = 1EC2H     ;$+7748
49A4:0081 A30000      MOV         WORD PTR 0000H,AX
```

If you use the VC command on the CALL instruction at address 49A4:006E

```
.VC 49A4:006E
```

the System Debug Monitor displays the following information.

```
S/W Int:  B8 (Nucleus)   entry code 0801     set exception handler
```

The "S/W Int: B8 (Nucleus)" means that the software interrupt number, "B8", identifies this call as a Nucleus call. The entry code within the Nucleus is "0801" which corresponds to an RQSETEXCEPTION$HANDLER system call.

Now suppose you want to see if the CALL instruction at 49A4:0071 is a system call. Type:

```
.VC 49A4:0071
```

The System Debug Monitor responds with the following message.

```
Not a system CALL
```

VC
EXAMPLES (continued)

If you use the VC command on the instruction at 49A4:0074, the System Debug Monitor responds with:

```
Not a CALL instruction
```

VD--Display a Job's Object Directory

The VD command displays a job's object directory.

PARAMETER

job token	The token for the job whose object directory you want to display.

DESCRIPTION

If the parameter is a valid job token, the System Debug Monitor displays the job's object directory as shown in Figure 4-2.

Directory size: xxxx		Entries used: xxxx
$name_1$	$token_1$	
$name_2$	tasks waiting	$token_2 \ldots token_i$
.	.	
.	.	
.	.	
$name_j$	$token_j$	
invalid entry		
$name_k$	$token_k$	
.	.	
.	.	
.	.	
$name_n$	$token_n$	

Figure 4-2. Format of VD Output

The fields in Figure 4-2 are as follows:

Directory size	The maximum allowable number of entries this job can have in its object directory.
Entries used	The number of entries used within the directory.
$name_1 \ldots name_n$	The names under which objects are cataloged.

VD
DESCRIPTION (continued)

$token_1 \ldots token_n$	Tokens for specific cataloged objects.
tasks waiting	Signifies tasks that have performed a RQ$LOOKUP$OBJECT on an object which has not been cataloged. The tokens following this field are the tokens for the tasks that are waiting.
invalid entry	This field appears only if the object directory has been destroyed or written over.

ERROR MESSAGES

The SDB returns the following error messages for the VD command

Error Message	Description
Syntax Error	You did not specify a parameter for the command or you made an error in entering the command.
TOKEN is not a Job	You entered a valid token that is not a job token.
*** INVALID TOKEN ***	You entered a value for the token that is not a valid token.

EXAMPLES

If you want to look at the object directory of job "528F," you would type:

```
.VD 528F
```

The System Debug Monitor responds as follows.

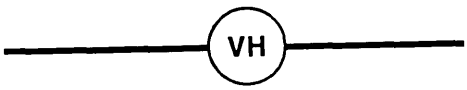
```
Directory size:  000A          Entries used:  0003

$                5229
R?IOUSER         5201
RQGLOBAL         528F
```

The words "$," "R?IOUSER," and "RQGLOBAL" are the names of the objects and their respective tokens are 5229, 5201, and 528F. There are no waiting tasks or invalid entries.

VH--Display Help Information

The VH command displays and describes the other 8 System Debug Commands.



PARAMETERS

There are no parameters for this call.

DESCRIPTION

The VH command lists all of the System Debug Commands along with their parameters and a short description of each command.

ERROR MESSAGE

The SDB returns the following error message for the VH command

Error Message	Description
Syntax Error	You made an error in entering the command.

EXAMPLE

If you type:

```
.VH
```

the System Debug Monitor responds as shown in Figure 4-3.

VH
EXAMPLE (continued)

```
iRMX 86 SYSTEM DEBUG MONITOR, Vx.x

   vc [<POINTER>]       Display system call,
                               (POINTER optional).
   vd <Job TOKEN>       Display job's object directory.
   vh                   Display help information.
   vj [<Job TOKEN>]     Display job heirarchy from specified level,
                               (TOKEN optional).
   vk                   Display ready and sleeping tasks.
   vo <Job TOKEN>       Display list of objects for specified job.
   vr <Seg TOKEN>       Display I/O Request/Result Segment.
   vs [<Count>]         Display stack and system call information,
                               (word count is in hex, default 10h).
   vt <TOKEN>           Display iRMX 86 object.
```

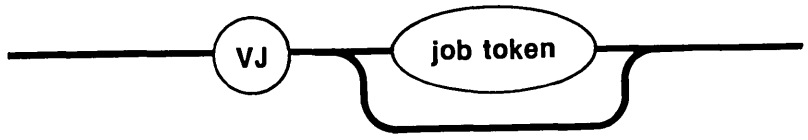
Figure 4-3. VH Display

NOTE

If you enter a zero (0) for any of the optional parameters in the previous commands, the SDB behaves as if you had used the default for that command.

VJ--Display the Job Hierarchy

The VJ command displays the job hierarchy from the level you specify.



PARAMETER

job token	The token for the job that heads the job heirarchy you want to display. If you do not specify a job token, VJ assumes the token to be the Root Job (default).

DESCRIPTION

The VJ command displays the token of the specified job and all the tokens of its offspring jobs. The offspring jobs are indented three spaces to show their position in the heirarchy. This command displays the job heirarchy as shown in Figure 4-4.

```
        iRMX/86 Job Tree

   token1
      token2
         token3
            token4
         token5
      token6
```

Figure 4-4. Format of VJ Output

The fields in Figure 4-4 are as follows:

$token_1$	The token for the root job or the job you specify.
$token_2 ... token_6$	The tokens for the offspring jobs of the root job or the job your specify.

VJ
DESCRIPTION (continued)

In Figure 4-4, tokens 2 and 6 are both indented three spaces symbolizing that they are child jobs of token 1. Token 1 has two offspring (6 and 2). Token 2 has two offspring (5 and 3). And token 3 has one offspring job (4). Your job may have more or less tokens, but the job heirarchy will be similar in format.

ERROR MESSAGES

The SDB returns the following error messages for the VJ command

Error Message	Description
Syntax Error	You made an error in entering the command.
TOKEN is not a Job	You entered a valid token that is not a job token.
*** INVALID TOKEN ***	You entered a value for the token that is not a valid token.
Error looking for root job	The SDB cannot find the root job.

EXAMPLES

If you want to examine the heirarchy of the root job you should type:

```
.VJ
```

Suppose the System Debug responds with the following job tree.

```
          iRMX/86 Job Tree

57DE
   528F
      51CE
         4F9F
   5741
   57B5
```

The previous display shows "57DE" to be the root job.

If you want to display the job tree from "51CE" you can type:

```
.VJ 51CE
```

VJ
EXAMPLES (continued)

The System Debug Monitor displays the following job tokens.

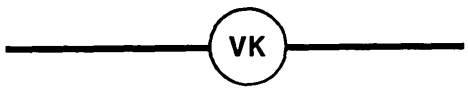
```
51CE
  4F9F
```

NOTE

The VJ command (without a parameter) needs the Nucleus interrupt vector and a small part of the Nucleus code in order to function correctly. If you destroy the Nucleus interrupt vector (by pressing the RESET switch) or if you write over the required part of Nucleus code, this command will not operate properly. You must re-initialize your system in order to restore the VJ command. See Chapter 3 for more information.

VK--Display Ready and Sleeping Tasks

The VK command displays the tokens for ready and sleeping tasks



PARAMETERS

This command has no parameters.

DESCRIPTION

The VK command displays the tokens for the tasks that are ready and asleep in the format shown in Figure 4-5.

```
Ready tasks:  xxxx
Sleeping tasks:
              xxxx
```

Figure 4-5. Format of VK Output

The fields in Figure 4-5 are as follows:

Ready tasks	The tokens for all the ready tasks in the system.
Sleeping tasks:	The tokens for all the sleeping tasks in the system.

ERROR MESSAGES

The System Debug Monitor uses the Nucleus interrupt vector and some Nucleus code in order to identify the ready and sleeping tasks. If you somehow destroy the Nucleus interrupt vector or the required code, the System Debug Monitor can't identify the ready and sleeping tasks. This causes the System Debug Monitor to display the following error messages.

```
Ready tasks:  Can't locate
Sleeping tasks:
              Can't locate
```

VK
ERROR MESSAGES (continued)

The most common reasons for this type of error are:

- pressing the RESET switch during debugging
- not initializing the Nucleus interrupt vector
- tasks writing over the Nucleus code
- tasks writing over iRMX 86 objects

The SDB also displays the following error message if you entered the command incorrectly.

Error Message	Description
Syntax Error	You made an error in entering the command.

EXAMPLE

If you want to display a list of all the ready and sleeping tasks in a the system, you can type:

```
.VK
```

In this example, the System Debug Monitor responds as follows:

```
Ready tasks:  4F02
Sleeping tasks:
              56F5   558A   56BF   5204   51B3   5090   55EC   5052
              5021   4FFE   5697   5238   511F   566E   563A   5769
              50D1   2302
```

NOTE

The VK command needs the Nucleus interrupt vector and a small part of the Nucleus code in order to function correctly. If you destroy the Nucleus interrupt vector (by pressing the RESET switch) or if you write over the required part of Nucleus code, this command will not operate properly. You must re-initialize your system in order to restore the VK command. See Chapter 3 for more information.

VO--Display Objects in a Job

The VO command displays the tokens of the objects in a job.

PARAMETER

job token	The token for the job whose objects you want to display.

DESCRIPTION

The VO command lists the tokens for a job's child jobs, tasks, mailboxes, semaphores, regions, segments, extensions, and composites in the format shown in Figure 4-6.

```
Child jobs:     xxxx    xxxx    xxxx...
Tasks:          xxxx    xxxx    xxxx...
Mailboxes:      xxxx    xxxx    xxxx...
Semaphores:     xxxx    xxxx    xxxx...
Regions:        xxxx    xxxx    xxxx...
Segments:       xxxx    xxxx    xxxx...
Extensions:     xxxx    xxxx    xxxx...
Composites:     xxxx    xxxx    xxxx...
```

Figure 4-6. Format of VO Output

The fields in Figure 4-6 are as follows:

Child jobs	The tokens for the child jobs within the job.
Tasks	The tokens for the tasks within the job.
Mailboxes	The tokens for the mailboxes within the job. A lower-case "o" immediately following a token for a mailbox means that one or more objects are queued at the mailbox. A lower-case "t" immediately following a token for a mailbox means that one or more tasks are queued at the mailbox.

VO
DESCRIPTION (continued)

Semaphores	The tokens for all the semaphores within the job. A lower-case "t" immediately following a token for a semaphore means that one or more tasks are queued at the semaphore.
Regions	The tokens for all the regions within the job. A lower-case "b" (busy) immediately following a token for a region means that a task is accessing information guarded by the region.
Segments	The tokens for all the segments within the job.
Extensions	The tokens for all the extensions within the job.
Composites	The tokens for all the composites within the job.

ERROR MESSAGES

The SDB returns the following error messages for the VO command

Error Message	Description
Syntax Error	You did not specify a parameter for the command or you made an error in entering the command.
TOKEN is not a Job	You entered a valid token that is not a job token.
*** INVALID TOKEN ***	You entered a value for the token that is not a valid token.

EXAMPLE

Suppose you want to look at the objects in "51CE."

```
.VO 51CE
```

The System Debug Monitor responds with the following display.

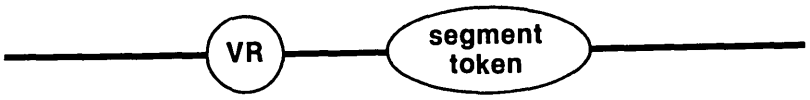
VO
EXAMPLE (continued)

Child jobs:	4F9F					
Tasks:	511F	50D1	5090	5052	5021	4FFE
Mailboxes:	5119	5110	5100 t	50FB t	50CE t	5089 t
Semaphores:	50FE	501F t				
Regions:						
Segments:	510C	5103	508C	504E	4FE6	4FCB
Extensions:						
Composites:	511C	5113	50C8	5083	4FF3	4FED

The previous display shows the tokens for the child jobs, tasks, mailboxes, semaphores, regions, segments, extensions, and composites in the job. It also tells you that there are tasks waiting at four mailboxes and one semaphore.

VR--Display I/O Request/Result Segment

The VR command displays information about an iRMX 86 Basic I/O System I/O request/result segment (IORS) associated with the segment token that you enter.



PARAMETER

Segment token	The token for a segment containing the IORS you want to display. This segment must be an IORS or the VR command returns invalid information.

DESCRIPTION

The VR command displays the names and values for the fields of a specific IORS. The SDB cannot determine if the segment contains a valid IORS so it is up to you to be sure that the segment does indeed contain an IORS. If the parameter is a valid segment token for an IORS, the SDB displays information about the IORS as shown in Figure 4-15. For more information concerning the following fields, see the GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O SYSTEMS.

I/O Request Result Segment

Status	xxxx	Unit status	xxxx
Device	xxxx	Unit	xx
Function	xxxxx	Subfunction	xxxxxxx
Count	xxxxxxx	Actual	xxxx
Device location	xxxxxxxx	Buffer pointer	xxxx:xxxx
Resp mailbox	xxxx	Aux pointer	xxxx:xxxx
Link forward	xxxx:xxxx	Link backward	xxxx:xxxx
Done	xxxx	Cancel ID	xxxx

Figure 4-7. Format of VR Output

VR
DESCRIPTION (continued)

The fields in Figure 4-7 are as follows:

Status — The exception code for the I/O operation.

Unit status — Additional status information. The Status field must be set to indicate the E$IO condition (002Bh) in order for the Unit Status to contain information. If the Status field is not an E$IO condition, the Unit Status field contains "N/A."

See the description of I/O Request/Result Segments in the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for further information.

Device — The number of the device for which the request is intended.

Unit — The number of the unit for which this request is intended.

Function — A description of the function for the operation to be performed by the Basic I/O System. The descriptions of the possible functions and their corresponding system calls are as follows:

Function	System Call
Read	RQAREAD
Write	RQAWRITE
Seek	RQASEEK
Special	RQASPECIAL
Att Dev	RQAPHYSICAL$ATTACH$DEVICE
Det Dev	RQAPHYSICAL$DETACH$DEVICE
Open	RQAOPEN
Close	RQACLOSE

If the function field can't be interpreted, the SDB displays the actual hexadecimal value of the field followed by a space and two question marks.

Subfunction — The actual function if the Function field contains "Special." The possible subfunctions are as follows:

Subfunction	Description
For/Que	Format or Query
Notify	Notify function
Satisfy	Stream file satisfy function

VR
DESCRIPTION (continued)

Subfunction (con't)	If the Function field doesn't contain "Special", then the Subfunction field will contain "N/A." If the Subfunction field can't be interpreted, the SDB displays the actual hexadecimal value of the field followed by a space and two question marks.
Count	The number of bytes of data that the Basic I/O System sets to be transferred.
Actual	The number of bytes of data actually transferred.
Device location	The 8-digit hexadecimal value of the absolute device location of the I/O operation.
Buffer pointer	The address of the buffer from which the Basic I/O System reads or to which it writes.
Resp mailbox	A token for the response mailbox. Upon completion of the operation, the device driver sends the IORS to this mailbox.
Aux pointer	The pointer to the location of any auxiliary data. The I/O System uses this data when the request is "Special" and the device driver needs extra data.
Link forward	The address of the next IORS in the request queue.
Link backward	The address of the previous IORS in the request queue.
Done	Indicates if the I/O operation is complete. A value of "TRUE" means that the I/O operation is complete, whereas a value of "FALSE" means that the I/O operation is incomplete.
Cancel ID	A word that identifies queued I/O requests that are to be removed from the queue by the CANCEL$IO procedure.

VR (continued)

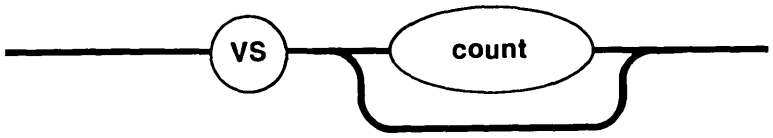
ERROR MESSAGES

The SDB returns the following error messages for the VR command

Error Message	Description
Syntax Error	You did not specify a parameter for the command or you made an error in entering the command.
TOKEN is not a Segment	You entered a valid token that is not a segment token.
*** INVALID TOKEN ***	You entered a value for the token that is not a valid token.

VS--Display Stack and System Call Information

The VS command identifies system calls (as does the VC command) and displays the stack.



PARAMETER

count	The count is the number of words (in hexadecimal) of the stack that you want to display. If you do not specify a count, VS uses a default value of 10H.

DESCRIPTION

The VS command identifies iRMX 86 system calls for all iRMX 86 subsystems (as does the VC command) and interprets the parameters on the stack. If one of these parameters is a string, the SDB disassembles the string. See the appropriate iRMX 86 Manual for additional information on system call information.

If you do not change the CS:IP value, the VS command interprets the CALL instruction at the current CS:IP. If you want to interpret a CALL instruction at a different CS:IP value, you must move the CS:IP to that value by using the iSBC 957B GO command.

The VS command uses the SS:SP (stack segment:stack pointer) registers to display the current values on the stack. If the instruction is an iRMX 86 system call, VS displays the system call and the stack information as shown in Figure 4-8.

```
xxxx:xxxx      xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   xxxx
xxxx:xxxx      xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   xxxx

S/W int:  xx  (subsystem)     entry code xxxx     system call

              :parameters.:
```

Figure 4-8. Format of VS Output

VS
DESCRIPTION (continued)

The fields in Figure 4-8 are as follows:

xxxx:xxxx	The contents of the SS:SP.
xxxx	Stack values.
parameters	The names of the stack values. The parameters correspond to the stack values directly above them.

The three remaining fields in Figure 4-8 are identical to those in the VC command.

S/W int: xx (subsystem)	The software interrupt number and the iRMX 86 subsystem that corresponds to that number.
entry code xxxx	The entry code for the system call within the subsystem.
system call	The name of the iRMX 86 system call.

ERROR MESSAGES

The SDB returns the following error messages for the VS command

Error Message	Description
Syntax Error	You made an error in entering the command.
Not a system CALL	The CS:IP is not pointing to a system call.
unknown entry code	This error message appears in place of the system call field when the SDB has been overwritten or when the SDB has misinterpreted the call.

If the instruction is not a CALL instruction, VS displays the contents of the words on the stack and no message. Even if you get the error messages previously described, the VS command displays the top of the stack.

EXAMPLES

Suppose you disassembled the following code using the iSBC 957B command, "DX".

VS
EXAMPLES (continued)

```
49A4:015C  E8CB1D        CALL          A = 1F1AH      ;$+7630
49A4:015F  B80000        MOV           AX,0
49A4:0162  50            PUSH          AX
49A4:0163  B80000        MOV           AX,0
49A4:0166  8EC0          MOV           ES,AX
49A4:0168  06            PUSH          ES
49A4:0169  50            PUSH          AX
49A4:016A  8D060800      LEA           AX,WORD PTR 0008H
49A4:016E  1E            CALL          DS
49A4:016F  50            PUSH          AX
```

Since the CS:IP is at 49A4:015C, when you use the VS command

```
.VS
```

the System Debug Monitor displays the following information.

```
4906:07CA      0008   4984   4EAC   4983   4983   0000   0600   4906
4906:07DA      49A4   0020   2581   4EAC   4EA1   4EE7   0000   0000

S/W int: B8 (Nucleus)   entry code  0301   delete mailbox

                :..excep$p..:.mbox.:
```

The parameters names in the previous example identify the stack values directly above them. The "excep$p" parameter signifies that the first two words represent a pointer (4984:0008) to the exception code. The "mbox" parameter signifies that the third word (4EAC) is the token for the mailbox being deleted.

Now, suppose you moved the CS:IP to 49A4:016E. If you invoke the VS command now, the debug monitor displays the stack as follows:

```
.VS

4906:07D0   4983   4983   0000   0600   4906   49A4   0020   2581
4906:07E0   F7C7   F7C7   F5C7   F5C7   F5C7   F5C7   F5CF   F5CF

Not a system CALL
```

The SDB displays the stack and a message which informs you that the call is not a system call.

If you want to display the stack at a call which happens to have more parameters than will fit on one line, the SDB automatically displays the extra parameters below the corresponding words in the stack. For example suppose you used the VS command and received the following display.

```
.VS
```

VS
EXAMPLES (continued)

```
   57CC:0F9A      015A   60C7   0000   60C6   60C6   0000   0600   57CC
   57CC:0FAA      60EF   0028   2322   0000   60C7   6618   6605   6623
   57CC:0FBA      6609   5A5F   5AF8   660B   0000   0000   0000   0000

   S/W Int:  B8 (Nucleus)  entry code 0100    create job

                :...excep$p...:t$flgs:stksze:..sp..:..ss..:..ds..:..ip..:
                :..cs..:.pri..:j$flgs:.exp$info$p..:maxpri:maxtsk:maxobj:
                :poolmx:poolmn:param.:dirsiz:
```

The previous display tells you that the CALL instruction is a Nucleus RQ$CREATE$JOB system call that has 18 parameters. The names of these parameters are shown between the colons (:). The words on the stack which correspond to the parameters are shown directly above the parameters.

The following displays show several examples of parameters with strings. The first line of each example is the CALL instruction and the lines following the VS command are the stack values.

```
   11B8:159D E8AA15          CALL      A = 2B4AH      ; $+5549

   .VS

   57CC:0F4E      0F8C   57CC   65FD   0000   6600   69A2   0000   6602
   57CC:0F5E      660B   3C13   6602   2325   66D2   0F7C   0DF7   FFFF

   S/W Int:  C0 (BIOS)  entry code 0002       attach file

                :...excep$p...:.mbox.:..subpath$p..:prefix:.user.:
  subpath-->07'example'
```

The previous display tells you that the CALL instruction is a Basic I/O System (BIOS) RQAATTACH$FILE system call that has 5 parameters. The "subpath$p" parameter points to a string that is 07 characters long. This string consists of the word "example."

```
    60EF:0776 E82714          CALL      A = 1BA0H      ; $+5162

    .VS

   57CC:0F98      014A   60C7   06A5   60EF   06A5   60EF   0000   0600
   57CC:0FA8      57CC   60EF   0028   2322   0000   60C7   000A   6605

   S/W Int:  C1 (EIOS)  entry code 0108       rename file

                :...excep$p...:.new$path$p..:...path$p....:
   new path-->04'XY70'
   path-->04'temp'
```

VS
EXAMPLES (continued)

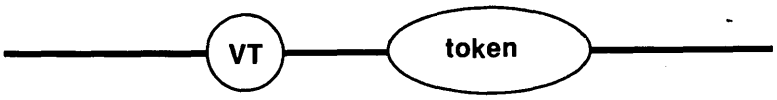
The previous display tells you that the CALL instruction is an Extended I/O System (EIOS) RQSRENAME$FILE system call that has 3 parameters. There are two parameters with strings in this example. The parameter, "new$path$p" points to a string that is 04 characters long. This string consists of the word "XY70." The parameter "path" is a string that is also 04 characters long and consists of the word "temp."

NOTE

If a string is longer than 50 characters, the SDB will display only the first 50 characters of the string.

VT--Display an iRMX 86 Object

The VT command displays information about the iRMX 86 object associated with the token you enter.



PARAMETER

token	The token for the object to be displayed.

DESCRIPTION

The VT command determines what type of object the token represents and displays information about that object. Both the information and the format in which the SDB displays the information are different for each type of object. The following sections are divided into display groups. Each display group contains the format and information for a particular type of object.

ERROR MESSAGES

The SDB returns the following error messages for the VT command

Error Message	Description
Syntax Error	You did not specify a parameter for the command or you made an error in entering the command.
*** INVALID TOKEN ***	You entered a value for the token that is not a valid token.

JOB DISPLAY

If the parameter is a valid job token, the SDB displays information about the job as shown in Figure 4-9.

VT
JOB DISPLAY (continued)

```
Object type = 1 Job

Current tasks  xxxx        Max tasks     xxxx        Max priority   xx
Current OBJs   xxxx        Max objects   xxxx        Parameter OBJ  xxxx
Directory size xxxx        Entries used  xxxx        Job flags      xxxx
Except handler xxxx:xxxx   Except mode   xx          Parent job     xxxx
Pool min       xxxx        Pool Max      xxxx        Initial size   xxxx
Pool size      xxxx        Allocated     xxxx        Largest seg    xxxx
```

Figure 4-9. Format of VT Output (Job Display)

The fields in Figure 4-9 are as follows:

Current tasks	The number of tasks currently existing in the job.
Max tasks	The maximum number of tasks that can exist in the job at the same time. This value was set when the job was created with the system call RQ$CREATE$JOB.
Max priority	The maximum (lowest numerically) priority allowed for any one task in the job. This value was set when the job was created with the system call RQ$CREATE$JOB.
Current OBJs	The number of objects currently existing in the job.
Max objects	The maximum number of objects that can exist in the job at the same time. This value was set when the job was created with the system call RQ$CREATE$JOB.
Parameter OBJ	The token for the object the parent job passed to this job. This value was set when the job was created with the system call RQ$CREATE$JOB.
Directory size	The maximum number of entries the job can have in its object directory. This value was set when the job was created with the system call RQ$CREATE$JOB.
Entries used	The number of objects currently cataloged in the object directory.

VT
JOB DISPLAY (continued)

Job flags	The job flags parameter that was specified when the job was created.
Except handler	The start address of the job's exception handler. This address was set when the job was created with the system call RQ$CREATE$JOB.
Except mode	The value that indicates when control is to be passed to the new job's exception handler. This value was set when the job was created with the system call RQ$CREATE$JOB.
Parent job	The token for the parent job of this job.
Pool min	The minimum size (in 16-byte paragraphs) of the job's memory. This value was set when the job was created.
Pool max	The maximum size (in 16-byte paragraphs) of the job's memory pool. This value was set when the job was created.
Initial size	The initial size (in 16-byte paragraphs) of the job's memory pool.
Pool size	The current size (in 16-byte paragraphs) of the job's memory pool.
Allocated	The number of 16-byte paragraphs in the job's memory pool which are currently allocated.
Largest Seg	The number of 16-byte paragraphs in the largest segment in the job's memory pool.

TASK DISPLAY

The SDB displays information about tasks in two different ways. The first display is for non-interrupt tasks and the second display is for interrupt tasks. The format of the two types of tasks is shown in Figures 4-10 and 4-11.

```
VT
TASK DISPLAY (continued)

object type = 2 Task

Static PRI       xx          Dynamic PRI      xx    Task state      xxxx
Suspend depth    xx          Delay req        xxxx  Last exchange   xxxx
Except handler   xxxx:xxxx   Except mode      xx    Task flags      xx
Containing job   xxxx        Interrupt task   no
```

Figure 4-10. Format of VT Output (Non-Interrupt Task)

```
object type = 2 Task

Static PRI       xx          Dynamic PRI      xx    Task state      xxxx
Suspend depth    xx          Delay req        xxxx  Last exchange   xxxx
Except handler   xxxx:xxxx   Except mode      xx    Task flags      xx
Containing job   xxxx        Interrupt task   yes   Int Level       xx
Master mask      xx          Slave mask       xx    Slave number    xx
Pending int      xx          Max interrupts   xx
```

Figure 4-11. Format of VT Output (Interrupt Task)

The fields in Figures 4-10 and 4-11 are as follows:

Static PRI — The current priority of the task. This value was set when the job was created with the system call RQ$CREATE$TASK.

Dynamic PRI — A temporary priority that the Nucleus sometimes assigns to the task in order to improve system performance.

Task state — The state of the task. There are five possible states:

State	Description
ready	ready for execution
asleep	task is asleep
susp	task is suspended
aslp/susp	task is both asleep and suspended
deleted	task is being deleted

If this field can't be interpreted, the SDB displays the actual haexadecimal value followed by a space and two question marks.

VT
TASK DISPLAY (continued)

Suspend depth	The current number of outstanding RQ$SUSPEND$TASK system calls applied to this task without corresponding RQ$RESUME$TASK system calls.
Delay req	The number of sleep units the task requested when it called RQ$SLEEP.
Last exchange	The token for the mailbox, region, or semaphore at which the task is currently waiting.
Except handler	The start address of the job's default exception handler. This value was set when the task was created with RQ$CREATE$TASK or when RQSETEXCEPTION$HANDLER was used.
Except mode	The value used to indicate when control is to be passed to the new task's exception handler. This value was set when the task was created with RQ$CREATE$TASK or when RQSETEXCEPTION$HANDLER was used.
Task flags	The task flags parameter used when the task was created with the system call RQ$CREATE$TASK.
Containing job	The token of the job which contains this task.
Interrupt task	"No" signifies that the task is not an interrupt task. In this case, there are no more fields in the display (see Figure 4-10). "Yes" signifies that the task is an interrupt task. In this case, there are six more fields in the display (see Figure 4-11).
Int level	The level that the interrupt task services. This level was set when the system call RQSETINTERRUPT was used.
Master mask	The hexadecimal value associated with the interrupt mask for the master interrupt controller. This value comes from the bits that correspond to the different master interrupt levels. Remember that bit numbers corresponds to interrupt level numbers. For example, bit 0 corresponds to interrupt level 0 and bit 7 corresponds to interrupt level 7. If the bit is set, the corresponding interrupt is disabled. For more information see the iRMX 86 NUCLEUS REFERENCE MANUAL.

VT
TASK DISPLAY (continued)

Slave mask	The hexadecimal value associated with the interrupt mask for a slave interrupt controller. This value comes from the bits that correspond to the different slave interrupt levels. Remember that bit numbers correspond to interrupt level numbers. For example, bit 0 corresponds to interrupt level 0 and bit 7 corresponds to interrupt level 7. If the bit is set, the corresponding interrupt is disabled. For more information see the iRMX 86 NUCLEUS REFERENCE MANUAL.
Slave number	The programmable interrupt controller number of the slave that is referred to by the slave mask. For more information see the iRMX 86 NUCLEUS REFERENCE MANUAL.
Pending int	The number of RQ$SIGNAL$INTERRUPT calls that are pending.
Max interrupts	The maximum number of RQ$SIGNAL$INTERRUPT calls that can be pending.

MAILBOX DISPLAY

The SDB displays information about mailboxes in three different ways. The first display appears when nothing is queued at the mailbox, the second display appears when tasks are queued at the mailbox, and the third display appears when objects are queued at the mailbox. The format for the three types of displays are shown in Figures 4-12, 4-13, and 4-14.

```
Object type = 3 Mailbox

Task queue head      xxxx              Object queue head     xxxx
Queue discipline     xxxx              Object cache depth    xxxx
Containing job       xxxx
```

Figure 4-12. Format of VT Output (Mailbox with No Queue)

VT
MAILBOX DISPLAY (continued)

```
Object type = 3 Mailbox

Task queue head      xxxx                Object queue head     xxxx
Queue discipline     xxxx                Object cache depth    xxxx
Containing job       xxxx

Task queue          xxxx    xxxx...
```

Figure 4-13. Format of VT Output (Mailbox with Task Queue)

```
Object type = 3 Mailbox

Task queue head      xxxx                Object queue head     xxxx
Queue discipline     xxxx                Object cache depth    xxxx
Containing job       xxxx

Object queue        xxxx    xxxx...
```

Figure 4-14. Format of VT Output (Mailbox with Object Queue)

The fields in Figure 4-12, 4-13, and 4-14 are as follows:

Task queue head	The token for the task at the head of the queue.
Object queue head	The token for the object at the head of the queue.
Queue discipline	The way you ordered the tasks and objects in the queue. The tasks and objects can be ordered in a "first-in/first-out" (FIFO) method or in a priority based method (PRI) when the mailbox is set up with RQ$CREATE$MAILBOX.
Object cache depth	The size of the high performance queue associated with the mailbox. The size of this cache was set up when the mailbox was created with RQ$CREATE$MAILBOX. When the list of tokens in the object queue is greater than the object cache depth, you have temporarily overflowed your high performance queue.

VT
MAILBOX DISPLAY (continued)

Containing job	The token for the job that contains this mailbox.
Task queue	A list of tokens for the tasks queued at the mailbox. This list appears in the display only if there are tasks queued at the mailbox.
Object queue	A list of tokens for the objects queued at the mailbox. This list appears in the display only if there are objects queued at the mailbox.

SEMAPHORE DISPLAY

The SDB displays information about semaphores in two ways. The first display appears when no tasks are queued at the semaphore, and the second display appears when tasks are queued at the semaphore. The formats for the two types of displays are shown in Figures 4-15 and 4-16.

```
Object type = 4 Semaphore

Task queue head     xxxx                Queue discipline    xxx
Current value       xxxx                Maximum value       xxxx
Containing job      xxxx
```

Figure 4-15. Format of VT Output (Semaphore with No Queue)

```
Object type = 4 Semaphore

Task queue head     xxxx                Queue discipline    xxx
Current value       xxxx                Maximum value       xxxx
Containing job      xxxx

Task queue          xxxx    xxxx...
```

Figure 4-16. Format of VT Output (Semaphore with Task Queue)

VT
SEMAPHORE DISPLAY (continued)

The fields in Figures 4-15 and 4-16 are as follows:

Task queue head	The token for the task at the head of the queue.
Queue discipline	The way the tasks are ordered in the queue. The tasks can be ordered in a "first-in/first-out" (FIFO) method or in a priority based method (PRI) when the semaphore is created with RQ$CREATE$SEMAPHORE.
Current value	The number of units currently contained in the semaphore.
Maximum value	The maximum number of units the semaphore can have. This number was set when the semaphore was created with RQ$CREATE$SEMAPHORE.
Containing job	The token for the job which contains the semaphore.
Task queue	A list of tokens that represent the tasks queued at the semaphore. This list appears in the display only if there are tasks queued at the semaphore.

REGION DISPLAY

If the parameter is a valid token for a region, the SDB displays the information about the region as shown in Figure 4-26.

```
Object type = 5 Region

Entered task       xxxx                    Queue discipline     xxxx
Containing job     xxxx
```

Figure 4-17. Format of VT Output (Region)

VT
REGION DISPLAY (continued)

The fields in Figure 4-17 are as follows:

Entered task	The token for the task that is currently accessing information guarded by the region.
Queue discipline	The way you ordered the tasks in the queue. The tasks can be ordered in a "first-in/first-out" (FIFO) method or in a priority based method (PRI) when the region is created with RQ$CREATE$REGION.
Containing job	The token for the job that contains the region.

SEGMENT DISPLAY

If the parameter is a valid token for a segment, the SDB displays the information about the segment as shown in Figure 4-18.

```
Object type = 6 segment

Num of paragraphs     xxxx              Containing job      xxxx
```

Figure 4-18. Format of VT Output (Segment)

The fields in Figure 4-18 are as follows:

Num of paragraphs	The number of 16-byte paragraphs in this segment. The size of the segment was specified when the system call, RQ$CREATE$SEGMENT was used.
Containing job	The token for the job that contains the region.

EXTENSION DISPLAY

If the parameter is a valid token for an extension, the SDB displays the information about the extension as shown in Figure 4-19.

```
VT
EXTENSION DISPLAY (continued)

Object type = 7 Extension

Extension type     xxxx                 Deletion mailbox     xxxxx
Containing job     xxxx
```

Figure 4-19. Format of VT Output (Extension)

The fields in Figure 4-19 are as follows:

Extension type	The type code associated with composite objects licensed by this extension. The type code was specified when the system call, RQ$CREATE$EXTENSION was used. See the iRMX 86 SYSTEM PROGRAMMERS REFERENCE MANUAL for more information.
Deletion mailbox	The token for the deletion mailbox associated with this extension. This mailbox was set up when the system call RQ$CREATE$EXTENSION was used.
Containing job	The token for the job that contains the extension.

COMPOSITE DISPLAY

There are five kinds of composite displays. Each display is an extension of the display that precedes it. The fields that are introduced in the figures are explained as they occur. The first kind appears for all composites except Basic I/O System (BIOS) composites. The second kind appears for BIOS user objects and the other three kinds appear for physical, for stream, and for named BIOS connections.

The format for the first kind of display is as shown in Figure 4-20.

```
Object type = 8 Composite

Extension type  xxxx    Extension OBJ    xxxx    Deletion mbox    xxxx
Containing job  xxxx    Num of entries   xxxx

Component list  xxxx    xxxx    xxxx    xxxx...
```

Figure 4-20. Format of VT Output (Composites Other Than BIOS)

VT
COMPOSITE DISPLAY (continued)

The fields in Figure 4-20 are as follows:

Extension type	The extension type code for the composite. This code was specified when the composite was created with RQ$CREATE$COMPOSITE.
Extension OBJ	The token for the extension object that represents the license to create this type of composite.
Deletion mbox	The token for the mailbox to which this composite goes when it is to be deleted. This mailbox was specified when the extension was created with RQ$CREATE$EXTENSION.
Containing job	The token for the job that contains the composite.
Num of entries	The number of component entries in the composite.
Component list	The list of tokens for the objects that currently make up the composite.

The format the for Basic I/O System user object display is shown in Figure 4-21.

```
Object type = 8 Composite

Extension type  xxxx   Extension OBJ    xxxx   Deletion mbox   xxxx
Containing job  xxxx   Num of entries   xxxx

     BIOS USER OBJECT:
User segment    xxxx   Number of IDs    xxxx

User ID list    xxxx    xxxx
```

Figure 4-21. Format of VT Output (BIOS User Object Composites)

The fields introduced in Figure 4-21 are as follows:

User segment	The token for the segment containing the user IDs.

VT
COMPOSITE DISPLAY (continued)

Number of IDs	The number of user IDs associated with the user ID.
User ID list	The list of user IDs.

The format for a connection to a physical file is shown in Figure 4-22.

```
Object type = 8 Composite

Extension type   xxxx  Extension OBJ    xxxx     Deletion mbox  xxxx
Containing job   xxxx  Num of entries   xxxx

      T$CONNECTION OBJECT
File driver   Physical  Conn flags      xx         Access          xxxx
Open mode     xxxx      Open share      xxxx       File pointer    xxxxxxxx
File node     xxxx      Device desc     xxxx       DUIB pointer    xxxx:xxxx
Num of conn   xxxx      Num of readers  xxxx       Num of writers  xxxx
File type     xxxx      File share      xxxx       File drivers    xxxx
Device gran   xxxx      Device size     xxxxxxxx   Device functs   xxxx
Device conn   xxxx      Device name     xxxx
```

Figure 4-22. Format of VT Output (Physical File Connection)

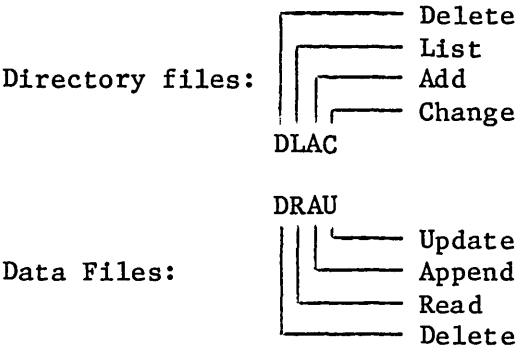
The fields introduced in Figure 4-22 are as follows:

File driver	The type of file driver to which this connection is attached. The three possible values are physical, stream, and named. If this field can't be interpreted, the SDB displays the actual hexadecimal value followed by a space and two question marks.
Conn flags	The flags for the connection. If bit 1 is set to one, this connection is active and can be opened; if bit 2 is set, this connection is a device connection.
Access	The access rights for this connection. The access rights are displayed in the same format as the display access rights for the DIR command in the Human Interface. This display uses a single character to represent a particular access right. If the file has the access right, the character appears. However, if the file does not have the

VT
COMPOSITE DISPLAY

Access (continued)

access right, a blank appears in the character position. The access rights along with the characters that represent them are as follows:



Open mode

The mode established when this connection was opened. The possible values are:

Open Mode	Description
Closed	Connection is closed
Read	Connection is open for reading
Write	Connection is open for writing
R/W	Connection is open for reading and writing

If this field can't be interpreted, the SDB displays the actual hexadecimal value followed by a space and two question marks. This value is set during a RQSOPEN or RQAOPEN system call.

Open share

The sharing status established when this connection was opened. The possible values are:

Share Mode	Description
Private	Private use only
Readers	File can be shared with readers
Writers	File can be shared with writers
ALL	File can be shared with all users

If this field can't be interpreted, the SDB displays the actual hexadecimal value followed by a space and two question marks. This value is set during a RQSOPEN or RQAOPEN system call. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information.

VT
COMPOSITE DISPLAY

File pointer — The current contents of the file pointer for this connection.

File node — A token to a segment that the Operating System uses to maintain information about the connection. The information in this segment appears in the next two fields.

Device desc — A token to the segment that contains the device descriptor. The device descriptor is used by the Operating System to maintain information about the connections to the device.

DUIB pointer — The address of the Device Unit Information Block (DUIB). See the GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O OPERATING SYSTEMS for more information on the DUIB.

Num of conn — The number of connections to the file.

Num of readers — The number of connections currently open for reading.

Num of writers — The number of connections currently open for writing.

File type — The type of file. This field is for Named files only so does not apply (N/A) to this display.

File share — The share mode of the file. This parameter defines how the file can be opened. The possible values are:

Share Mode	Description
Private	Private use only
Readers	File can be shared with readers
Writers	File can be shared with writers
ALL	File can be shared with all users

If this field can't be interpreted, the SDB displays the actual hexadecimal value followed by a space and two question marks. This value is set during RQSOPEN or RQAOPEN system calls. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information.

File drivers — The file drivers that can be used with the containing the file.

VT
COMPOSITE DISPLAY

Bit	Driver
0	Physical file
1	Stream file
2	reserved
3	Named file

Device gran — The granularity (in bytes) of the device.

Device size — The size (in bytes) of the device.

Device functs — describes the functions supported by the device on which this file resides. Each bit has a corresponding function. If that bit is set, then the corresponding function is supported by the device.

Bit	Function
0	F$READ
1	F$WRITE
2	F$SEEK
3	F$SPECIAL
4	F$ATTACH$DEV
5	F$DETACH$DEV
6	F$OPEN
7	F$CLOSE

Device conn — The number of connections to the device.

Device name — The 14-character name of the device where this file resides.

The format for a stream connection display is shown in Figure 4-23.

```
Object type = 8 Composite

Extension type  xxxx      Extension OBJ  xxxx      Deletion mbx  xxxx
Containing job  xxxx      Num of entries xxxx

     T$CONNECTION OBJECT
File driver  Stream   Conn flags      xx        Access          xxxx
Open mode    xxxx     Open share      xxxx      File pointer    xxxxxxxx
File node    xxxx     Device desc     xxxx      DUIB pointer    xxxx:xxxx
Num of conn  xxxx     Num of readers  xxxx      Num of writers  xxxx
File type    xxxx     File share      xxxx      File drivers    xxxx
Device gran  xxxx     Device size     xxxxxxxx  Device functs   xxxx
Device conn  xxxx     Device name     xxxx
Req queued   xxxx     Queued con      xxxx      Open conn       xxxx
```

Figure 4-23. Format of VT Output (Stream File Connections)

VT
COMPOSITE DISPLAY (continued)

The fields introduced in Figure 4-23 are as follows:

Req queued	The number of requests that are currently queued at the stream file. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information.
Queued conn	The number of connections that are currently queued at the stream file. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information.
Open conn	The number of connections that are currently open on the stream file. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information.

The format for a named file connection display is shown in Figure 4-24.

```
Object type = 8 Composite

Extension type  xxxx    Extension OBJ   xxxx    Deletion mbx    xxxx
Containing job  xxxx    Num of entries  xxxx

      T$CONNECTION OBJECT
File driver     Named     Conn flags      xx        Access          xxxx
Open mode       xxxx      Open share      xxxx      File pointer    xxxxxxxx
File node       xxxx      Device desc     xxxx      DUIB pointer    xxxx:xxxx
Num of conn     xxxx      Num of readers  xxxx      Num of writers  xxxx
File type       xxxx      File share      xxxx      File drivers    xxxx
Device gran     xxxx      Device size     xxxxxxxx  Device functs   xxxx
Device conn     xxxx      Device name     xxxx
Fnode           xxxx      Fnode flags     xxxx      Owner           xxxx
File ID         xxxx      File gran       xxxx      Fnode PTR(s)    xxxx:xxxx
Total blocks    xxxxxxxx  Total size      xxxxxxxx  This size       xxxxxxxx
Volume gran     xxxx      Volume size     xxxxxxxx  Volume name     86/330
```

Figure 4-24. Format of VT Output (Named File Connections)

The fields introduced in Figure 4-24 are as follows:

File type — The type of file. The possible values are:

File type	Description
DIR	Directory file
DATA	Data file

VT
COMPOSITE DISPLAY (continued)

Fnode — A token for the segment in which the Basic I/O System keeps a copy of the fnode. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about fnodes.

Fnode flags — A word containing flag bits. Each bit has a corresponding description. If that bit is one, then the corresponding description is true; if the bit is zero, then the corresponding description is false.

Bit	Description
0	This fnode is allocated
1	The file is a long file
2	Primary fnode
3-4	Reserved
5	This file has been modified
6	This file is marked for deletion
7-15	reserved

Owner — The ID of the owner of the file. If this field has a value of FFFF, then the field is interpreted as "World." See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.

File ID — The number of the file's file descriptor. The file descriptor is a Basic I/O System data structure containing file attribute and status data.

File gran — The granularity of the file (in device granularity units).

Fnode PTR(s) — The values of the fnode pointers. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.

Total blocks — The total number of volume blocks currently used for the file; this includes indirect blocks. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.

Total size — The total size (in bytes) of the file; this includes actual data only. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.

This size — The size (in bytes) of the file. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.

VT
COMPOSITE DISPLAY (continued)

Volume gran	The granularity (in bytes) of the volume.
Volume size	The size (in bytes) of the volume.
Volume name	The name of the volume.

INDEX

REQUEST FOR READER'S COMMENTS

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

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

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

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

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

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

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

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS . . .

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

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

O.M.S. Technical Publications

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

Printed in U.S.A.