iRMX™ 86 SYSTEM DEBUGGER REFERENCE MANUAL

CONTENTS

FIGURES

***

CHAPTER 1
ORGANIZATION

This manual contains four chapters. Some of the chapters contain introductory or overview material that you might not need to read if you are already familiar with the iSBC 957B, iSDM 86, or iSDM 286 monitor. Other chapters contain reference material that you can use as you debug your system. You can use this chapter to determine which of the other chapters you should read.

The remaining chapters of the manual are the following:

Chapter 2	This chapter describes the features of the System Debugger and its relationship to the other tools for debugging iRMX 86 applications. You should read this chapter if you are going through the manual for the first time.
Chapter 3	This chapter gives a variety of facts pertaining to the use of the System Debugger. You should read this chapter if you are installing the System Debugger and/or configuring it into your system.
Chapter 4	This chapter contains detailed descriptions of the System Debugger 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 commands.

***

CHAPTER 2
INTRODUCTION

The development of almost every system requires debugging. To aid you in the development of iRMX 86-based application systems, Intel provides the iRMX 86 Debugger, the ICE-86 In-Circuit Emulator, the iSDM 86 and iSDM 286 System Debug Monitors, and the iSBC 957B monitor. The System Debugger extends the capabilities of the three monitors. This manual describes the System Debugger extension. The iRMX 86 DEBUGGER REFERENCE MANUAL describes the iRMX 86 Debugger. The USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE describes the iSBC 957B monitor. The iSDM 86 SYSTEM DEBUG MONITOR REFERENCE MANUAL describes the iSDM 86 monitor. And the iSDM 286 SYSTEM DEBUG MONITOR REFERENCE MANUAL describes the iSDM 286 monitor. The following sections describe the relative advantages of the various debugging tools.

ADVANTAGES OF THE iRMX™ 86 DEBUGGER

The iRMX 86 Debugger is a debugging tool that is "sensitive" to the data structures that the Nucleus maintains. The iRMX 86 Debugger allows you to:

- Manipulate or examine any task while other tasks in the system continue to run. This distinguishes the iRMX 86 Debugger from the iRMX 86 System Debugger, which requires that the application system be "frozen."
- Monitor system activity without interfering with execution.
- Examine and interpret data structures that are associated with the Nucleus and the Nucleus objects.

ADVANTAGES OF THE ICE®-86 EMULATOR

The ICE-86 emulator provides in-circuit emulation for iAPX 86, 88 microprocessor-based systems, meaning that it "stands in" for the 8086 or 8088 microprocessor in your target iRMX 86-based system during development. The ICE-86 emulator allows you to:

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

ADVANTAGES OF THE iSBC® 957B, iSDM™ 86, AND iSDM™ 286 MONITORS

The iSBC 957B, iSDM 86, and iSDM 286 monitors each support both interactive commands and system I/O routines. Each allows you to:

- Disassemble code.
- Set execution and memory breakpoints.
- Display memory.

ADVANTAGES OF THE iRMX™ 86 SYSTEM DEBUGGER

You can extend the capabilities of the iSBC 957B, iSDM 86, or iSDM 286 monitor the System Debugger part of your operating system. In addition to retaining the features of the monitors, the System Debugger:

- Identifies and interprets iRMX 86 system calls.
- Displays iRMX 86 objects.
- Examines the stack of a task to determine which iRMX 86 system calls it has made recently.

REQUIREMENTS OF THE iRMX™ 86 SYSTEM DEBUGGER

In order to use the System Debugger, you must have exactly one of the following hardware configurations, with whatever support hardware that is required (independent of the System Debugger):

- A terminal connected directly to an iAPX 86-, 88-, 186-, 188-, or 286-based board.

or

- An Intellec system connected to an iAPX 86-, 88-, 186-, 188-, or 286-based board.

You must also have:

- The monitor portion of the iSBC 957B iAPX 86, 88 Interface and Execution Package or the iSDM 86 or iSDM 286 System Debug Monitor.
- At least the minimal configuration of the Nucleus. The System Debugger needs only a small portion of valid Nucleus code, so most of the System Debugger commands will function even if you accidentally write over part of the Nucleus.

See the next chapter for more information about configuring and installing the System Debugger.

***

CHAPTER 3
USING THE SYSTEM DEBUGGER

This chapter contains various facts about using the iRMX 86 System Debugger.

HOW THE SYSTEM DEBUGGER IS SUPPLIED

The System Debugger is supplied as a file along with the other parts of the iRMX 86 Operating System.

USE RESTRICTIONS OF THE SYSTEM DEBUGGER

One of the capabilities of the System Debugger is that it can display information about specific invocations of system calls. However, it can do this correctly only for applications that use the PL/M-86 SMALL, COMPACT, or LARGE model of segmentation.

CONFIGURING THE SYSTEM DEBUGGER

To use the System Debugger to debug your application, you must configure it into the application. You do this simply by responding to two prompts that the iRMX 86 Interactive Configuration Utility issues. One of the prompts asks whether you want the System Debugger to be part of your system. The other, which applies only if you respond affirmatively to the first prompt, asks which interrupt level you want to use to invoke the System Debugger manually.

INVOKING THE SYSTEM DEBUGGER

There are two ways of invoking the System Debugger. As the previous section implies, one way is to press the button that is physically tied to the interrupt level you specify during configuration. The other way, which requires that your system include the Human Interface, is to use the DEBUG command.

The DEBUG command syntax requires the pathname of a loadable file. DEBUG loads the indicated file and then passes control to the iSBC 957B, iSDM 86, or iSDM 286 monitor. Normally, the file the DEBUG command loads is the file that is to be debugged. However, in this case the file to be debugged (the application system incorporating the System Debugger) is already in memory. To satisfy the requirement that the DEBUG command load some file, but without corrupting your application, specify the pathname of a file that the DEBUG command can load harmlessly into an area of memory not used by the application. A file you can use for this purpose is the TIME command of the Human Interface. It requires little memory and, when loaded, is automatically located where it does not interfere with the application.

See the iRMX 86 OPERATOR'S MANUAL for more information concerning the DEBUG command.

After the DEBUG command loads the file into memory or after you press the interrupt button, the monitor issues its period (.) prompt, and you can begin entering System Debugger commands. These commands are the subject of the next chapter.

RETURNING TO YOUR APPLICATION

When you have finished debugging your application system, you can start it up again by means of the go (G) command of the monitor.

***

CHAPTER 4
COMMANDS

This chapter contains detailed descriptions of the iRMX 86 System Debugger commands, in alphabetical order. There is also a Command Dictionary that lists the commands in functional groups.

This chapter uses "CS:IP" to mean "code segment:instruction pointer." The chapter also contains several examples of System Debugger commands entered at the terminal. In the examples, user input is underscored to distinguish it from System Debugger output. Carriage returns are not shown after the user input but they are required for the System Debugger to execute the command.

CHECKING VALIDITY OF TOKENS

The iRMX 86 Operating System maintains tokens in doubly-linked lists. Whenever you enter a command that requires a token as a parameter, the System Debugger 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, VT, and VU commands as well as the tokens that are listed in the displays.

If one of a token's links is bad, the System Debugger generates an error message along with the information the command that you entered usually displays. The token you enter as a parameter of the System Debugger command always appears in each line as the center value in the display of tokens. The displays for forward- and backward-link errors are as follows:

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

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

Arrows to the right indicate forward links and those to the left indicate backward links. A question mark appearing before or after a value signals a forward or backward link error.

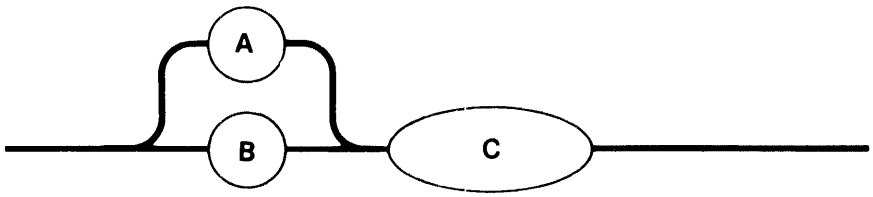
If both links are bad, the System Debugger considers the token invalid and displays the following message:

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

The presence of a link error means that iRMX 86 data structures have been corrupted. The most common reason for this problem is overwriting. One of your tasks might have accidentally written over part of the system data structures and/or code. If you are using the non-maskable interrupt, another possible cause of a link error is that you interrupted the Nucleus while it was setting up the links. If either of these things happen, you must re-initialize the System Debugger (and perhaps your System). Only then can you use the VD, VJ, VK, VO, VR, VT, and VU commands without getting another link error. See Chapter 3 in this manual for more information about initializing the System Debugger on your system.

PICTORIAL REPRESENTATION OF SYNTAX

This manual uses a schematic device to illustrate the syntax of commands. The schematic consists of what looks like an aerial view of a model railroad, with syntactic elements 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 doing so consists of the syntactic elements that it encounters on its journey. The following pictorial syntax shows two valid sequences: AC and BC.



x-455

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

x-456

you can enter:

```
.VR xxxx
```

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

Even though all syntax diagrams show uppercase letters, such as VR, entering lowercase equivalents of those letters produces the same effect.

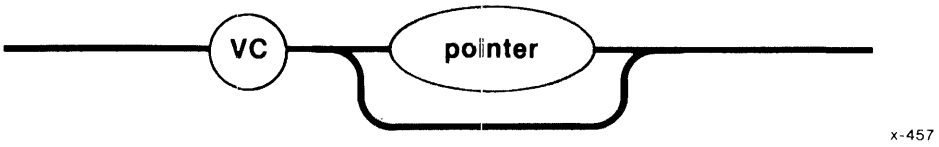
DISPLAY OF NUMERICAL VALUES

In all of the displays that this chapter discusses, all numerical values are given in hexadecimal form.

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 optional pointer parameter can be any valid iSBC 957B, iSDM 86, or iSDM 286 address. The System Debugger uses this address as the address of the CALL instruction to be checked.

If you do not supply a pointer, the System Debug Monitor uses the default pointer, which is the current CS:IP. If you specify an IP value but not a CS value, the System Debugger uses the current CS as the default base.

DESCRIPTION

If the CALL instruction is an iRMX 86 system call, the VC command 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 defined 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.

NOTE

The System Debugger uses the software interrupt number associated with the displayed entry code to determine whether the CALL instruction represents a system call. It is possible, but not likely, that the System Debugger can interpret a sequence of bytes as a software interrupt (INT) instruction and then (inaccurately) reported that a CALL instruction is an iRMX 86 system call.

ERROR MESSAGES

The System Debugger 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 points to a CALL instruction that is not an iRMX 86 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 DX command of the iSBC 957, iSDM 86, or iSDM 286 monitor:

```
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, that is, you enter:

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

the System Debugger responds by displaying 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. Enter:

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

The System Debugger responds with the following message.

```
Not a system CALL
```

Finally, if you use the VC command on the instruction at 49A4:0074, the System Debugger responds with:

```
Not a CALL instruction
```

VD--Display A Job's Object Directory

The VD command displays a job's object directory.

x-458

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 Debugger displays the job's object directory, as shown in Figure 4-2.

```
Directory size:    xxxx            Entries used:    xxxx

name1              token1
name2              tasks waiting   token2...tokeni
.                  .
.                  .
.                  .
namej              tokenj
invalid entry
namek              tokenk
.                  .
.                  .
.                  .
namen              tokenn
```

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 presently in the directory.
$name_1 \ldots name_n$	The names under which objects are cataloged.
$token_1 \ldots token_n$	Tokens for the cataloged objects.
tasks waiting	Signifies that one or more tasks have performed an RQ$LOOKUP$OBJECT on an object that is not cataloged. The tokens following this field identify the tasks that are still waiting for the objects to be cataloged.
invalid entry	This field appears only if the specified job's object directory has been destroyed or written over.

ERROR MESSAGES

The System Debugger 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 ***	The value you entered for the token is not a valid token.

EXAMPLES

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

```
.VD 528F
```

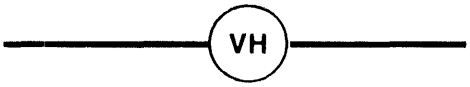
The System Debugger responds as follows:

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

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

The symbols "$," "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 ten System Debugger commands.



x-459

PARAMETERS

There are no parameters for this call.

DESCRIPTION

The VH command lists all of the System Debugger commands, along with their parameters and a short description of each command.

ERROR MESSAGE

The System Debugger 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 enter:

```
.VH
```

the System Debugger responds as shown in Figure 4-3, where the brackets indicate optional parameters.

```
iRMX 86 SYSTEM DEBUGGER, Vx.y

   vc [<POINTER>]       Display system call,
   vd <Job TOKEN>       Display job's object directory.
   vh                   Display help information.
   vj [<Job TOKEN>]     Display job hierarchy from specified level.
   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.
   vt <TOKEN>           Display iRMX 86 object.
   vu <Task TOKEN>      Display system calls on stack of specified task.
```

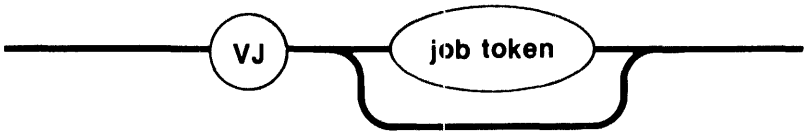
Figure 4-3. VH Display

NOTE

If you use zero (0) for any of the optional parameters shown in Figure 4-3, the effect is the same as if you omitted the parameter altogether.

VJ--Display The Job Hierarchy

The VJ command displays the portion of the job hierarchy that descends from the level you specify.



PARAMETER

job token — The token for the job whose descendant job hierarchy you want to display.

If you do not specify a job token, VJ assumes the default job, which is the root job.

The specified job, whether it is specified explicitly or whether it is the default (root) job, should not have more than 44 generations of job descendants. Otherwise, the display of the excessively-long branch is discontinued, an error message is displayed, and the System Debugger prompts for another command.

DESCRIPTION

The VJ command displays the token of the specified job and all the tokens of its descendant jobs. It also displays the tokens of the jobs (and their descendants) at the same level as the specified job. The descendant jobs are indented three spaces to show their position in the hierarchy. This command displays the job hierarchy 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 descendant jobs of the root job or the job you specify.

In Figure 4-4, jobs 2 and 6 are both indented three spaces to signify that they are children of job 1. Similarly, jobs 3 and 5 are depicted as children of job 2, and job 4 is shown as the child of job 3.

ERROR MESSAGES

The System Debugger 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 ***	The value you entered for the token is not a valid token.
Error looking for root job	The System Debugger cannot find the root job.
SDB job nest limit exceeded	The job specified in the command invocation (or the default job) has more than 44 generations of job descendants.

EXAMPLES

If you want to examine the hierarchy of the root job, enter:

```
.VJ
```

Suppose the System Debugger responds with the following job tree.

```
            iRMX/86 Job Tree

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

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

If you want to display the descendant jobs of "51CE", enter:

```
.VJ 51CE
```

The System Debugger displays the following job tokens:

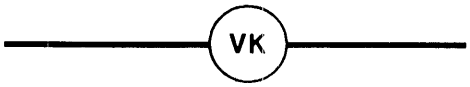
```
51CE
   4F9F
```

NOTE

The VJ command (without a parameter) requires 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 does not operate properly. You must re-initialize your system in order to restore the VJ command.

VK--Display Ready And Sleeping Tasks

The VK command displays the tokens for the tasks that are in the ready and sleeping states.



x-461

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

ERROR MESSAGES

The System Debugger returns the following error messages for the VK command:

Error Messages	Description
Ready tasks: Can't locate Sleeping tasks: Can't locate	The system is corrupted. See the following explanation.
Syntax error	You made an error in entering the command.

The System Debugger 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 Debugger can't identify the ready and sleeping tasks.

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.

If any of these problems apply to your system, you must re-initialize your system.

EXAMPLE

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

```
.VK
```

In this example, the System Debugger 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
```

VO--Display Objects In A Job

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

x-462

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 specified job's offspring jobs.
Tasks	The tokens for the tasks in the specified job.

Mailboxes	The tokens for the mailboxes within the job. A lower-case "o" immediately following a mailbox token means that one or more objects are queued at the mailbox. A lower-case "t" immediately following a mailbox token means that one or more tasks are queued at the mailbox.
Semaphores	The tokens for the semaphores in the specified job. A lower-case "t" immediately following a semaphore token means that one or more tasks are queued at the semaphore.
Regions	The tokens for the regions in the specified job. A lower-case "b" (busy) immediately following a region token means that a task has access to information guarded by the region.
Segments	The tokens for the segments in the specified job.
Extensions	The tokens for the extensions in the specified job.
Composites	The tokens for the composites in the specified job.

ERROR MESSAGES

The System Debugger 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 ***	The value you entered for the token is not a valid token.

EXAMPLE

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

```
.VO 51CE
```

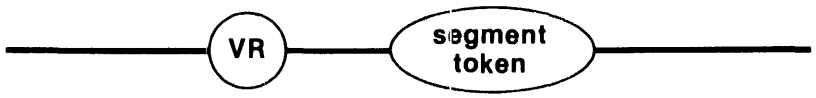
The System Debugger responds with the following display:

```
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 at one semaphore.

VR--Display I/O Request/Result Segment

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



x-463

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 System Debugger cannot determine whether the segment contains a valid IORS, so it is up to you to ensure that the segment does indeed contain an IORS. If the parameter is a valid segment token for a segment containing an IORS, the System Debugger displays information about the IORS as shown in Figure 4-7.

The contents of the IORS pertain to the most recent I/O operation in which this IORS was used. 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
Connection token	xxxx		

Figure 4-7. Format Of VR Output

The fields in Figure 4-7 are as follows:

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

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

Unit status — Additional status information. The contents of this field are meaningful only when the Status field is set to the E$IO condition (002BH).

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 last request was intended.

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

Function — The operation that was performed by the Basic I/O System. The possible functions 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 contains an invalid value, the System Debugger displays the value in this field, followed by a space and two question marks.

Subfunction — A further specification of the function that applies only when the Function field contains "Special." The possible subfunctions and their descriptions are as follows:

Subfunction (con't)

Subfunction	Description
For/Que	Format or Query
Satisfy	Stream file satisfy function
Notify	Notify function
Device char	Device characteristics
Get Term Attr	Get terminal attributes
Set Term Attr	Set terminal attributes
Signal	Signal function
Rewind	Rewind tape
Read File Mark	Read file mark on tape
Write File Mark	Write file mark on tape
Retention Tape	Take up slack on tape

If the Function field doesn't contain "Special", then the Subfunction field contains "N/A." If the Subfunction field contains an invalid value, the System Debugger displays the value of the field followed by a space and two question marks.

Count — The number of bytes of data called for in the I/O request.

Actual — The number of bytes of data transferred in response to the request.

Device location — The eight-digit hexadecimal address of the byte where the I/O operation began on the specified device.

Buffer pointer — The address of the buffer from which the Basic I/O System read or to which it wrote in response to the request.

Resp mailbox — A token for the response mailbox to which the device sent the IORS after the operation.

Aux pointer — The pointer to the location of auxiliary data, if any. This field is significant only when the Function field contains "Special."

Link forward — The address of the next IORS in the queue where the IORS waited to be processed.

Link backward — The address of the previous IORS in the queue where the IORS waited to be processed.

Done — This field is always present but applies only to IORS's for I/O operations on random-access devices. When applicable, it indicates whether the I/O operation has been completed. The possible values are TRUE (FFFFH) and FALSE (0000H).

Cancel ID	A word that is used by device drivers to identify I/O requests that need to be cancelled. A value of 0 indicates a request that cannot be cancelled.
Connection token	The token for the file connection that was used to issue the request for the I/O operation.

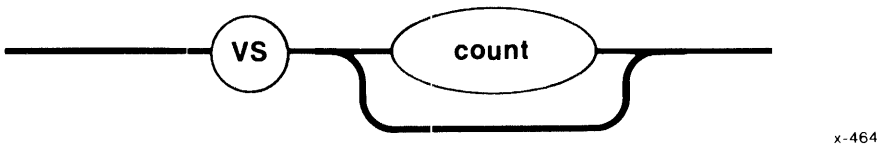
ERROR MESSAGES

The System Debugger 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 ***	The value you entered for the token is not a valid token.
Segment wrong size, not an IORS	The specified segment is neither four nor five paragraphs in length, so it is not an I/O request/result segment.

VS--Display Stack And System Call Information

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



PARAMETER

count — A decimal or hexadecimal value that specifies the number of words from the stack that are to be included in the display. A suffix of T, as in 16T, means decimal. No suffix or a suffix of H indicates hexadecimal.

If you do not specify a count, VS assumes the default value, 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 a parameter is a string, the System Debugger displays the string. See the appropriate iRMX 86 manual for additional information about system calls.

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, iSDM 86, or iSDM 286 GO command.

The VS command uses current values of the SS:SP (stack segment:stack pointer) registers to display the current stack values. 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

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 System Debugger always displays the words at the top of the stack. If it encounters problems, it then returns one of the following error messages.

Error Message	Description
Syntax Error	You made an error in entering the command.
Not a system CALL	The CS:IP is pointing to a CALL instruction that is not a system call.

Unknown entry code	This message indicates that one of two infrequent events has occurred. One is that the System Debugger has mistaken an operand for the software interrupt (INT) instruction. The other possibility is that a software link from user code into iRMX 86 code has been corrupted.

If the instruction is not a CALL instruction, VS displays the contents of the words on the stack and no message.

EXAMPLES

Suppose that by some means, such as the X command of the iSBC 957B, iSDM 86, or iSDM 286 monitor, you determine that the SS:SP is 4906:07CA. Suppose further that you then you use the VS command, as follows:

```
.VS

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

Now, suppose that you move the SS:SP to 4906:07D0. 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 System Debugger displays the stack and a message which informs you that the instruction is a CALL instruction but is not a system call.

When an iRMX 86 system call is executed, its parameters are pushed onto the current stack, and then a CALL instruction is issued with the appropriate address. If you want to display the stack at such a call when there are more parameters than will fit on one line, the System Debugger automatically displays multiple lines of words from the stack, with corresponding lines of parameter description directly below them.

For example, suppose that you use the VS command as follows:

```
.VS

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:
```

This display indicates that the CALL instruction is a Nucleus RQ$CREATE$JOB system call having 18 parameters. The names of these parameters are shown between the colons (:). As usual, the words on the stack correspond to the parameters shown directly below those words.

The following display indicates that the CALL instruction is a Basic I/O System (BIOS) RQAATTACH$FILE system call having five parameters. The "subpath$p" parameter points to a string that is seven characters long. This string consists of the word "example."

```
.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 following display indicates that the CALL instruction at CS:IP is an Extended I/O System RQSRENAME$FILE system call having three parameters. There are two parameters with strings in this example. The new$path$p parameter points to a string that is four characters long. This string contains "XY70." The path$p parameter points to a string that is also four characters long and contains "temp."

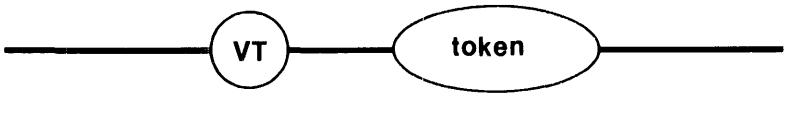
```
.VS

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'
```

NOTE

If a string is more than 50 characters in length, the System Debugger will display only the first 50 characters of the string. And if the pointer to a string is 0000:0000, the System Debugger does not display 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 for which the System Debugger will display information.

DESCRIPTION

The VT command ascertains the type of object represented by the token and displays information about that object. Both the information and the format in which the System Debugger displays the information depend entirely upon the type of the object. The following sections are divided into display groups. Each display group illustrates the format of the display for a particular type of object.

ERROR MESSAGES

The System Debugger 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 ***	The value you entered for the token is not a valid token.

JOB DISPLAY

If the parameter that you supply is a valid job token, the System Debugger displays information about the job that has that token, as Figure 4-9 shows.

```
Object type = 1 Job

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

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.
Parent job	The token for the job's parent.
Pool min	The minimum size (in 16-byte paragraphs) of the job's memory pool. 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 currently-allocated 16-byte paragraphs in the job's memory pool.
Largest Seg	The number of 16-byte paragraphs in the largest contiguous portion of the job's memory pool.

TASK DISPLAY

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

```
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      Kernel saved ss:sp  xxxx:xxxx
```

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      Kernel saved ss:sp  xxxx:xxxx
```

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 task's containing job was created.

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. The five possible states, as they are displayed, are:

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 contains an invalid value, the System Debugger displays the value followed by a space and two question marks.

Suspend depth — The number of RQ$SUSPEND$TASK system calls that have been applied to this task without corresponding RQ$RESUME$TASK system calls.

Delay req — The number of sleep units the task requested when it last specified a delay at a mailbox or semaphore, or when it last called RQ$SLEEP. If the task has not done any of these things, this field contains 0.

Last exchange — The token for the mailbox, region, or semaphore at which the task most recently began to wait.

Except handler	The start address of the job's default exception handler. This value was set either when the task was created, by means of RQ$CREATE$TASK or RQ$CREATE$JOB, or later by means of RQSETEXCEPTION$HANDLER.
Except mode	The value used to indicate the exceptional conditions under which control is to be passed to the new task's exception handler. This value was set either when the task was created, by means of RQ$CREATE$TASK or RQ$CREATE$JOB, or later by means of RQSETEXCEPTION$HANDLER.
Task flags	The task flags parameter used when the task was created with the RQ$CREATE$TASK system call.
Containing job	The token of the job that contains this task.
Interrupt task	"No" signifies that the task is not an interrupt task. In this case, there are no fields following this field in the display. (See Figure 4-10.) "Yes" signifies that the task is an interrupt task. In this case, there are additional fields in the display. (See Figure 4-11.)
Kernel saved ss:sp	The contents of the ss:sp registers when the task last left the ready state.
Int level	The level that the interrupt task services. This level was set when this task called RQSETINTERRUPT.
Master mask	The value associated with the interrupt mask for the master interrupt controller. This value represents the master interrupt levels that are disabled by virtue of the interrupt level that the task services. For example, if the task services interrupt level 51 (in octal), then master levels 6 and 7 are disabled, so the master mask field is 11000000B (=C0H). For more information concerning interrupt levels, see the iRMX 86 NUCLEUS REFERENCE MANUAL.

Slave mask	The value associated with the interrupt mask for a slave interrupt controller. This value represents the slave interrupt levels that are disabled by virtue of the level that the task services. For example, if the task services interrupt level 51 (octal), then slave levels 2 through 7 are disabled, so the slave level field is 11111100B (=FCH). For more information concerning interrupt levels, 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. This value depends entirely upon the interrupt level that the task services. If the value in the Int level field (after conversion to octal) is xy, then y+1 is the value in this field.
Pending int	The number of RQ$SIGNAL$INTERRUPT calls that are pending for this level.
Max interrupts	The maximum number of RQ$SIGNAL$INTERRUPT calls that can be pending for this level.

MAILBOX DISPLAY

The System Debugger displays information about mailboxes in three different ways. The first display appears when nothing is queued at the mailbox, the second appears when tasks are queued at the mailbox, and the third appears when objects are queued at the mailbox. The formats of the three types of display 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)

```
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. If the task queue for this mailbox is empty, this field contains 0.
Object queue head	The token for the object at the head of the queue. If the object queue for this mailbox is empty, this field contains 0.
Queue discipline	The manner in which tasks are queued at the mailbox. Tasks are queued "first-in/first-out" (FIFO) or by priority (PRI), depending upon how the mailbox was specified to RQ$CREATE$MAILBOX.
Object cache depth	The size of the high performance portion of the object queue that is associated with the mailbox. This size was specified when the mailbox was created with RQ$CREATE$MAILBOX.
Containing job	The token for the job that contains this mailbox.

Task queue	A list of tokens for the tasks queued at the mailbox in the order in which tasks are queued. If no tasks are queued at the mailbox, this list does not appear.
Object queue	A list of tokens for the objects queued at the mailbox in the order in which the objects are queued. If no objects are queued at the mailbox, this list does not appear.

SEMAPHORE DISPLAY

The System Debugger displays information about semaphores in two ways. The first display appears when no tasks are queued at the semaphore, and the second 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)

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 manner in which tasks are queued at the semaphore. The tasks are queued "first-in/first-out" (FIFO) or by priority (PRI), depending upon how the semaphore was specified when created with RQ$CREATE$SEMAPHORE.
Current value	The number of units currently held by the semaphore.
Maximum value	The maximum number of units the semaphore can hold. This number was specified when the semaphore was created with RQ$CREATE$SEMAPHORE.
Containing job	The token for the job that contains the semaphore.
Task queue	A list of tokens for the tasks queued at the semaphore, in the order in which the tasks are queued there. If no tasks are queued at the semaphore, this list does not appear.

REGION DISPLAY

If the parameter that you supply is a valid token for a region, the System Debugger displays information about the associated region as shown in Figure 4-26.

```
Object type = 5 Region

Entered task       xxxx                    Queue discipline     xxxx
Containing job     xxxx

Task queue         xxxx    xxxx  ...
```

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

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 manner in which tasks are queued at the region. The tasks are queued first-in/first-out (FIFO) or by priority (PRI), depending upon how the region was specified when created with RQ$CREATE$REGION.

Containing job	The token for the job that contains the region.
Task queue	Tokens for the tasks waiting to gain access to data guarded by the region. This line is displayed only if a task already has access to the data guarded by the region.

SEGMENT DISPLAY

If the parameter that you supply is a valid token for a segment, the System Debugger displays information about the associated 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 segment was created with the RQ$CREATE$SEGMENT system call.
Containing job	The token for the job that contains the segment.

EXTENSION OBJECT DISPLAY

If the parameter that you supply is a valid token for an extension, the System Debugger displays information about the associated extension as shown in Figure 4-19.

```
Object type = 7 Extension

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

Figure 4-19. Format Of VT Output (Extension Object)

The fields in Figure 4-19 are as follows:

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

COMPOSITE OBJECT DISPLAY

There are five kinds of composite displays. The first kind depicts all composites except those defined in the Basic I/O System (BIOS). The second kind depicts BIOS user objects. The remaining kinds depict BIOS physical, stream, and named file connections.

The format for the display of non-BIOS objects 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 (Composite Object Other Than BIOS)

The fields in Figure 4-20 are as follows:

Extension type	The code for the extension type of the extension object used to create this composite. This code was specified when the extension object was created with RQ$CREATE$EXTENSION.
Extension obj	The token for the extension object used to create this composite object.

Deletion mbox	The token for the mailbox to which this composite goes when the composite 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 object.
Num of entries	The number of component entries in the composite object.
Component list	The list of tokens for the components of the composite.

The format for the 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 Composite)

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

User segment	The token for the segment containing the user IDs for the user object.
Number of IDs	The number of user IDs associated with the user object.
User ID list	List of the user IDs associated with the user object.

The format for a (file) 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
IORS cache       xxxx     File node       xxxx     Device desc    xxxx
Dynamic DUIB     xxxxx    DUIB pointer    xxxx:xxxx Num of conn   xxxx
Num of readers   xxxx     Num of writers  xxxx     File share     xxxxxxx
File drivers     xxxx     Device gran     xxxx     Device size    xxxxxxxx
Device functs    xxxx     Num dev conn    xxxx     Device name    xxxxxxxxxx
```

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

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

File driver — The BIOS file driver to which this connection is attached. The three possible values are physical, stream, and named. If this field contains an invalid value, the System Debugger displays the 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 to one, this is a device connection. (Bit 0 is the low-order bit.)

Access — The access rights for this connection. 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 access right, a hyphen (-) appears in the character position. The access rights, along with the characters that represent them, are as follows:

```
Directory files:  |------- Delete
                  | |------ List
                  | | |----- Add
                  | | | |---- Change
                  DLAC

                  DRAU
                  | | | |---- Update
Data Files:       | | | |----- Append
                  | | |------ Read
                  | |------- Delete
```

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 contains an invalid value, the System Debugger displays the value, followed by a space and two question marks. If this value is Read, Write, or R/W, this value was specified when the connection was opened.

Open share

The sharing status established for this connection when it was opened. The sharing status for a connection is a subset of the sharing status of the file (see the File share field). 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 the connection is not open, then 0 is displayed. If this field contains an invalid value, the System Debugger displays the value, followed by a space and two question marks. This probably indicates that the connection data structure has been corrupted.

File pointer

The current location of the file pointer for this connection.

IORS cache

The token for the segment at the head of the BIOS list of used IORS's. These IORS's are being saved for the RQ$WAIT$IO system call to use again. The list is empty if 0000 appears in this field.

File node

The token for 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

The token for 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.

Dynamic DUIB — Indicates whether a DUIB was created dynamically when the device associated with this connection was attached.

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

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

Num of readers — The number of connections to the file that are currently open for reading.

Num of writers — The number of connections to the file that are currently open for writing.

File share — The share mode of the file. This parameter defines how other connections to the file can be opened. The share mode of a file is a superset of the sharing status of each of the connections to the file (see the Open share field). 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 contains an invalid value, the System Debugger displays the value, followed by a space and two question marks. This probably means that the internal data structure for the file or the fnode for the file has been corrupted. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information about sharing modes for files and connections.

File drivers — The file drivers that the file can be connected to. If the file can be connected to a file driver, then the bit in the display is set to 1. Bit 0 is the rightmost bit.

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

Device gran	The granularity (in bytes) of the device. This is the minimum number of bytes that can be written to or read from the device in a single (physical) I/O operation.
Device size	The capacity (in bytes) of the device.
Device functs	Describes the functions supported by the device on which this file resides. Each bit in the low-order byte of the field corresponds to one of the possible device functions. If that bit is set to 1, 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

Num dev conn	The number of connections to the device.
Device name	The 14- (or fewer) character name of the device where this file resides.

The format for a (file) connection to a stream file 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
IORS cache      xxxx    File node       xxxx        Device desc   xxxx
Dynamic DUIB    xxxxx   DUIB pointer    xxxx:xxxx   Num of conn   xxxx
Num of readers  xxxx    Num of writers  xxxx        File share    xxxxxxx
File drivers    xxxx    Device gran     xxxx        Device size   xxxxxxxx
Device functs   xxxx    Num dev conn    xxxx        Device name   xxxxxxxxxx
Req queued      xxxx    Queued conn     xxxx        Open conn     xxxx
```

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

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

Req queued	The number of requests that are currently queued at the stream file.
Queued conn	The number of connections that are currently queued at the stream file.
Open conn	The number of connections to the stream file that are currently open.

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
IORS cache      xxxx     File node       xxxx       Device desc     xxxx
Dynamic DUIB    xxxxx    DUIB pointer    xxxx:xxxx  Num of conn     xxxx
Num of readers  xxxx     Num of writers  xxxx       File share      xxxx
File drivers    xxxx     Device gran     xxxx       Device size     xxxxxxxx
Device functs   xxxx     Num dev conn    xxxx       Device name     xxxx
Num of buffers  xxxx     Fixed update    xxxx       Update timeout  xxxx
Fnode number    xxxx     File type       xxxx       Fnode flags     xxxx
Owner           xxxxx    File/Vol gran   xxxx       Fnode PTR(s)    xxxx:xxxx
Total blocks    xxxxxxxx Total size      xxxxxxxx   This size       xxxxxxxx
Volume gran     xxxx     Volume size     xxxxxxxx   Volume name     xxxxxx
```

Figure 4-24. Format Of VT Output (Named File Connection)

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

Num of buffers	The number of buffers allocated for blocking and unblocking I/O requests involving the device. A value of 0 indicates that the device is not a random-access device.
Fixed update	Indicates whether the device uses the fixed update feature. For more information about fixed updating, see the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL.

Update timeout

The length of the timeout for the update timeout feature, measured in Nucleus time units. For more information about update timeout, see the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL.

Fnode number

The fnode number of this file. For more information about fnodes, see the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL.

File type

The type of named file. The possible values are:

File type	Description
DIR	Directory file
DATA	Data file
SPACEMAP	Volume free space map file
FNODEMAP	Free fnodes map file
BADBLOCKMAP	Bad blocks file

Fnode flags

A word containing flag bits. If a bit is set to 1, the following description applies. Otherwise, the description does not apply. (Bit 0 is the low-order bit.)

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 BASIC I/O SYSTEM REFERENCE MANUAL for more information about owners of files.

File/Vol gran

The granularity of the file (in volume granularity units).

Fnode PTR(s)

The values of the fnode pointers. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about fnode pointers.

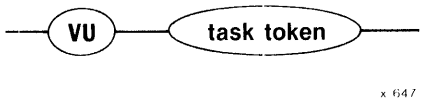
Total blocks

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 about blocks.

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 total number of bytes allocated to the file for data. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
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.

VU--Display The System Calls In A Task's Stack

The VU command displays (unwinds) the iRMX 86 system calls in the stack of the task whose token you enter.



x 647

PARAMETER

token	The token for the task whose stack is to be searched for system calls.

DESCRIPTION

The VU command accepts a token for a task and then searches the task's stack for iRMX 86 system calls, starting at the top of the stack. For each system call it finds there, it displays two things:

- The address of the next instruction to be executed on behalf of the task after the system call has finished running. This is the return address for the call.
- The VS display with two lines of stack values (or more if required for parameters), as if the CALL instruction for the system call were in the CS:IP register and the displayed stack values were at the top of the stack.

This command requires that the task stack reside inside an iRMX 86 segment.

The VU command uses internal iRMX 86 data structures to get some of its information. Immediately after the system call at the top of the task's stack runs to completion, the data structures are updated. Therefore, there is a brief moment when the information the VU command uses is obsolete. This means that it is possible, although unlikely, that the first system call the VU command displays is not valid.

Figure 4-25 illustrates the format of one system call display by the VU command. System calls can be nested, with one calling another, so some invocations of the VU command produce multiple displays of the type shown in the figure. The example that follows illustrates this.

If there are no system calls in the stack of the indicated task, the VU command displays the message:

```
No system calls on stack
```

```
Return cs:ip - yyyy:yyyy
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 4-25. Format Of VU Output

The fields in Figure 4-25 are as follows:

Return cs:ip	The return address for the system call of this display.
xxxx:xxxx	The address of the (top of the) portion of the stack that is devoted to this call.
xxxx	Stack values.
parameters	The parameter names associated with the stack values. The parameters correspond to the stack values directly above them.
S/W int: xx (subsystem)	The software interrupt number for this call and the iRMX 86 subsystem corresponding 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 System Debugger returns the following error messages for the VU command.

Error Message	Description
Syntax Error	You made an error in entering the command.
*** INVALID TASK TOKEN ***	The value you entered for the token is not a valid task token.
Stack not in an iRMX 86 segment	The stack of the indicated task is not in an iRMX 86 segment, as is required.

EXAMPLE

This example shows how the VU command responds when system calls are nested. The task for the example has called RQSWRITE$MOVE of the Extended I/O System. RQSWRITE$MOVE has called RQ$A$WRITE of the Basic I/O System. And RQAWRITE has called RQ$RECEIVE$MESSAGE to wait for the data transfer to be completed.

Before the message arrives signalling the completion of the transfer, the VU command is invoked, as follows:

```
.VU ????
```

The System Debugger responds by displaying the following:

```
Return cs:ip - 0104:576A
416A:01B2     01C8   416A   01CC   416A   FFFF   376E   8763   2988
416A:01C2     1550   0000   719E   2FF9   3440   E55E   5000   DD54

S/W Int: B8 (Nucleus)  Entry code  0303    Receive message

            :...excep$p...:....resp$p...:.time.:.mbox.:

Return cs:ip - 1756:08E7
416A:01D4     01EA   416A   3F56   0400   0000   42E9   429A   7866
416A:01E4     1430   5246   01FE   22F9   1400   021D   0000   01FE

S/W Int: C0 (BIOS)  Entry code  000E    Write

            :...excep$p...:.mbox.:.count:..buffer$p...:.conn.:

Return cs:ip - 3A98:06FA
416A:0218     0020   39F4   0400   0034   39F4   429A   5A84   9344
416A:0228     4456   0000   0000   C7C7   C7C7   C7C7   C7C7   C7C7

S/W Int: C1 (EIOS)  Entry code  0101    Write move

            :...excep$p...:.count:..buffer$p...:.conn.:
```


***