intel

iRMX® I Dynamic Debugger Reference Manual

Order Number: 462929-001

intel®

iRMX® I
Dynamic Debugger Reference Manual

Order Number: 462929-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

In locations outside the United States, obtain additional copies of Intel documentation by contacting your local Intel sales office. For your convenience, international sales office addresses are located directly after the reader reply card in the back of the manual.

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 or to keep current the information contained in this document.

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

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

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

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

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

XENIX, MS-DOS, Multiplan, and Microsoft are trademarks of Microsoft Corporation. UNIX is a trademark of Bell Laboratories. Ethernet is a trademark of Xerox Corporation. Centronics is a trademark of Centronics Data Computer Corporation. Chassis Trak is a trademark of General Devices Company, Inc. VAX and VMS are trademarks of Digital Equipment Corporation. Smartmodem 1200 and Hayes are trademarks of Hayes Microcomputer Products, Inc. IBM, PC/XT, and PC/AT are registered trademarks of International Business Machines. Soft-Scope is a registered trademark of Concurrent Sciences.

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

PREFACE

This manual documents the Dynamic Debugger, a subsystem of the iRMX® I Operating System that allows you to examine your application system interactively to find and correct errors. It contains introductory and overview material, as well as detailed descriptions of all Dynamic Debugger commands.

READER LEVEL

This manual is intended for both application and system programmers who are familiar with the concepts and terminology introduced in the *iRMX® I Nucleus User's Guide*.

CONVENTIONS

The Debugger commands are listed alphabetically in Chapter 4. The first occurrence of each command name is printed in blue ink and appears on the outside upper corner of the page; subsequent occurrences are printed in black ink.

This manual contains several examples of Debugger commands entered at a terminal. In these examples, entries made from your terminal appear in boldface type (**this is an example of boldface type.**)

User input appears in one of the following forms:

```
as blue text
```

```
as bolded text within a screen
```

RELATED PUBLICATIONS

The following manuals provide additional background and reference information.

Manual	Number
Introduction to the iRMX® Operating System	462909-001
iRMX® I Nucleus User's Guide	462925-001
iRMX® I Terminal Handler Reference Manual	462930-001
Guide to the iRMX® I Interactive Configuration Utility	462923-001

CONTENTS

Chapter 1. Introduction

Chapter 2. Special Characters

Chapter 3. Command Syntax

Chapter 4. Debugger Commands

Chapter 5. Configuration

Appendix A. Error Messages

Index

Figures

1 INTRODUCTION

1.1 OVERVIEW OF iRMX® I OPERATING SYSTEM DEBUGGING TOOLS

The development of almost every software application requires debugging. To aid in the development of iRMX I-based systems, Intel provides various debugging tools. This section gives an overview of some of the debugging tools available from Intel for iRMX I-based systems.

1.1.1 iRMX® I Dynamic Debugger

The first tool available for system debugging is the iRMX I Dynamic Debugger. The Dynamic Debugger will be referred to as the "iRMX I Debugger" or "the Debugger" for the remainder of this manual. The Debugger enables you to dynamically examine the data structures handled by the iRMX I operating system. For example, the Debugger can show which tasks are waiting at a particular mailbox while the application program is running, enabling you to easily debug a multitasking operation.

The Debugger supplies its own Terminal Handler, which includes all of the capabilities described in the *iRMX® I Terminal Handler Reference Manual.* Your application software can use the Debugger's Terminal Handler in its standard form, or you can configure your own version (or versions) of it. (Refer to the *Guide to the iRMX® I Interactive Configuration Utility* for further configuration information.)

1.1.2 System Debug Monitors

The second tool available to the programmer is the Intel series of monitors. This group of monitors includes the iSBC® 957B monitor, the iRMX I System Debugger (SDB), and the iSDM™ monitor. All of these monitors can, among other functions, single step instruction code, set execution and memory breakpoints, display memory in various formats (such as ASCII), perform I/O read and write operations, and move, search, and compare blocks of memory. The SDB extends the use of the other monitors so you can directly examine operating system data structures. For more information on the monitors, consult the following manuals: *iSDM™ System Debug Monitor Reference Manual,* or the *iRMX® System Debugger Reference Manual.*

1.2 iRMX® I DEBUGGER IMPLEMENTATION ON 80186, 80286 and 386™ CPUs

One of the advantages of the iRMX I Operating System is that it can run on any one of several Intel microprocessors. Thus, the use of the iRMX I Debugger does not change even though the microprocessor running the operating system may be the 8086, 80186, 8088, 80188, 80286, or the 386™ CPU. This is because the Debugger acts on those features, such as registers, that the 8086, 80186, 8088, 80188, 80286 and 386™ microprocessors have in common. Thus, the iRMX I Debugger will appear to see an 8086 CPU although another microprocessor may be physically running the system. (The 80286/386 microprocessors achieve this by running in the Real Address mode.)

This same principle of compatibility applies to the 8087, 80287 and 387™ Numeric Processor Extension (NPX) used by the particular microprocessors. The iRMX I Debugger will see only those registers the 8087 NPX has in common with the other Numeric Processor Extensions (e.g., the 80286/20 processor).

1.3 OVERVIEW OF THE CAPABILITIES OF THE iRMX® I DEBUGGER

The iRMX I Debugger enables you to:

- Use the Debugger as a task, job, or system exception handler.
- View iRMX I object lists, including the lists of jobs, tasks, ready tasks, suspended tasks, asleep tasks, task queues at exchanges, object queues at mailboxes, exchanges, and iRMX I segments.
- Inspect jobs, tasks, exchanges, segments, composites, and extensions.
- Examine and/or alter the contents of absolute memory locations.
- Set, change, view, and delete breakpoints.
- View the list of tasks that have incurred breakpoints and remove tasks from it.
- Declare a task to be the breakpoint task.
- Examine and/or alter the breakpoint task's register values.
- Set, change, view, and delete special variables for debugging.

1.4 INVOKING THE DEBUGGER

You can invoke the Debugger from your iRMX I terminal by entering

```
CONTROL-D
```

The Debugger responds with its sign-on message:

```
iRMX I DEBUGGER <version no.>
Copyright <years> Intel Corporation - All Rights Reserved
*
```

The asterisk is the prompt character for the Debugger and indicates the Debugger is ready to accept input.

Besides the functions the Debugger can execute when you invoke it, there are two services it can execute at any time, even when not invoked.

If a task encounters a breakpoint, the Debugger responds as described in Chapter 4.

If a task has the Debugger as its exception handler and the task causes an exceptional condition, then the Debugger displays a message to that effect at the terminal. A task can get the Debugger as its exception handler in one of the following ways:

- By using the SET$EXCEPTION$HANDLER system call.
- By acquiring the Debugger as the default exception handler. This is done during configuration. Refer to the *iRMX® I Interactive Configuration Utility Reference Manual* for a description of this process.
- By having the Debugger declared as the exception handler when the task is created with CREATE$JOB. For example, this code sets up one of these calls:

```
RQ$DEBUGGER$EX: PROCEDURE (EX$CODE, PARAM$NO, RESERVED,
                           NDP$STATUS, DUMMY$IF$COMPACT) EXTERNAL;
    DECLARE
        EX$CODE             WORD,
        PARAM$NO            BYTE,
        RESERVED            WORD,
        NDP$STATUS          WORD,
        DUMMY$IF$COMPACT    WORD;
END RQ$DEBUGGER$EX;

    DECLARE EXCEPT$BLOCK STRUCTURE (
        EXCEPT$PROC         POINTER,
        EXCEPT$MODE         BYTE);
        .
        .
```

```
        .

EXCEPT$BLOCK.EXCEPT$PROC = @RQ$DEBUGGER$EX;
EXCEPT$BLOCK.EXCEPT$MODE = ZERO$ONE$TWO$OR$THREE;
        .
        .
        .

RQ$CREATE$JOB(...,@EXCEPT$BLOCK,...);
```

For this code to work, the task code must be linked to the CROOT.LIB library supplied with the Nucleus. The DUMMYIFCOMPACT parameter in the RQ$DEBUGGER$EX declaration is a dummy parameter that you must include if the task is compiled using the PL/M-86 COMPACT.

SPECIAL CHARACTERS 2

2.1 INTRODUCTION TO SPECIAL CHARACTERS

In addition to the Debugger commands listed in Chapter 4, the Debugger recognizes several special characters. This chapter lists these characters and describes their functions.

2.2 END-OF-LINE CHARACTERS

The Debugger takes input one line at a time from its Terminal Handler. The end-of-line characters separate these individual input lines. The Debugger recognizes three end-of-line characters:

CARRIAGE RETURN
LINE FEED
ESCAPE

Both CARRIAGE RETURN and LINE FEED send the current input line to the Debugger for processing. ESCAPE discards the current input line and displays a prompt.

2.3 CONTROL-S

The Debugger displays information on the terminal by sending output messages to its Terminal Handler. Application tasks can also send messages to the same terminal. To suppress output from application tasks during a debugging session, type CONTROL-S. The Debugger then stores the output from application tasks until you type CONTROL-Q. If you do not enter CONTROL-S, any output from tasks is interspersed with output from the Debugger. CONTROL-S has no effect on output from the Debugger.

2.4 CONTROL-Q

CONTROL-Q negates the effect of a previously entered CONTROL-S character. To resume the output from tasks, type CONTROL-Q. CONTROL-Q also causes the Debugger to display all output that was suppressed by CONTROL-S. CONTROL-Q has no effect on output from the Debugger.

2.5 CONTROL-O

Certain Debugger command responses are lengthy and can roll off the screen. To freeze the top part of such a display before it disappears, enter CONTROL-O. This discards all output (including Debugger prompts) until you enter another CONTROL-O. The discarded output cannot be retrieved.

2.6 CONTROL-D

Occasionally you may want to terminate a Debugger memory command function response before it completes. For example, if you asked for a display of memory locations 0000H to 0FFFFH, you may change your mind because of the length of the display. To abort the display and regain the Debugger prompt, enter CONTROL-D.

Note that CONTROL-O affects the display only, while CONTROL-D stops the function entirely.

COMMAND SYNTAX 3

3.1. INTRODUCTION TO COMMAND SYNTAX

When using the iRMX I Debugger, you sit at a terminal and type commands. This chapter describes the syntactical standards for commands to the Debugger and introduces notational conventions used in this manual.

3.2 CONVENTIONS

The first one or two characters of a command constitute a key sequence for the command:

- Most Debugger commands are specified by one or two letters (e.g., BL, BT, D, DB, G, I, L, M, N, Q, R, V, and Z).
- In a few cases, a command is specified by a name plus a letter or letters.. A name consists of a period followed by a variable name.

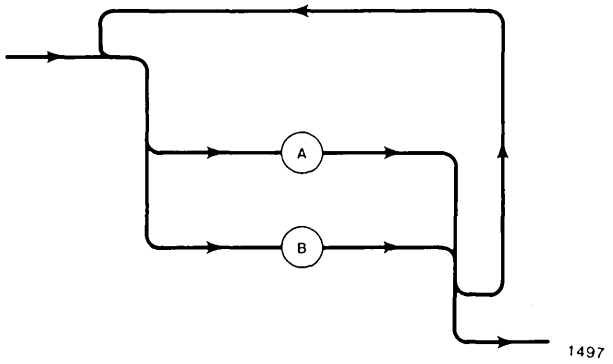
After the key initial sequence, a command may be followed by one or more parameters or additional specifiers. Blanks are used between elements of a command; they are required except as follows:

- Immediately after a command key that is not a name.
- Between a letter or digit and a non-letter or non-digit. Legal non-digit/letter characters are the following:

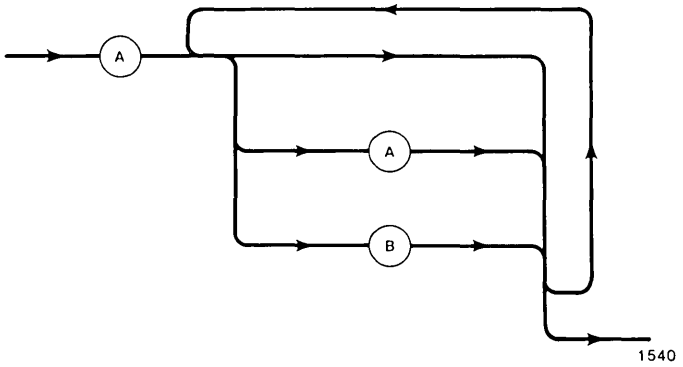
```
;   @   =   /   :   (   )   *   +   -   ,
```

3.3 PICTORIAL REPRESENTATION OF SYNTAX

In this manual, a schematic device illustrates the syntax of commands. The schematic consists of what looks like an aerial view of a model railroad, with syntactic entities scattered along the track. Imagine that a train enters the system at the upper left, drives around as much as it can or wants to (sharp turns and backing up are not allowed), and finally departs at the lower right. The command it generates in so doing consists, in order, of the syntactic entities that it encounters on its journey. For example, a string of A's and B's, in any order, would be depicted as



If such a string has to begin with an A, the schematic could be drawn as:



In the second drawing, A must appear twice because it is playing two roles: it is a mandatory first symbol and an optional symbol that may be used after the first symbol. Note that a train could avoid the second A but cannot avoid the first A. The arrows are not necessary and henceforth are omitted.

3.4 SPECIAL SYMBOLS FOR THE DEBUGGER

Instead of A and B, the following are used in the rest of the manual:

- CONSTANT. Constants are always hexadecimal. Unlike such constants in PL/M-86, a trailing H is optional. Leading zeros are not necessary unless they help to distinguish between constants and other parts of the command. For example, AH is a register in the 8086, but 0AH is a constant.
- NAME. A name is a period followed by up to 11 characters, the first of which must be alphabetic. The other characters can be alphabetic, numeric, question marks (?), or dollar signs ($).

  Examples:

```
.task

.mailbox$7
```

- ITEM. An item is either an expression or one of the segment registers of the CPU. The values of items are used variously as tokens and as offsets in Debugger commands. Graphically, an item is defined in Figure 3-1.

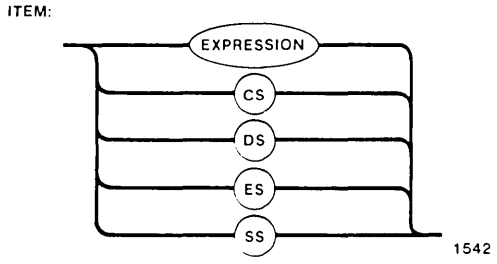


Figure 3-1. Syntax Diagram for Item

- EXPRESSION. As in algebra, an expression is either a term or the result of adding and subtracting terms. Also as in algebra, a term is a product; each factor in the product is a constant, a name, a parenthetical expression, or one of the registers AX, BX, CX, DX, DS, ES, SS, CS, IP, FL, SI, DI, BP, and SP. Graphically, term and expression are shown in Figure 3-2.

NOTE

If the computed value of an expression is too large to fit into four hexadecimal digits, then only the low order four digits are used (overflow is ignored).

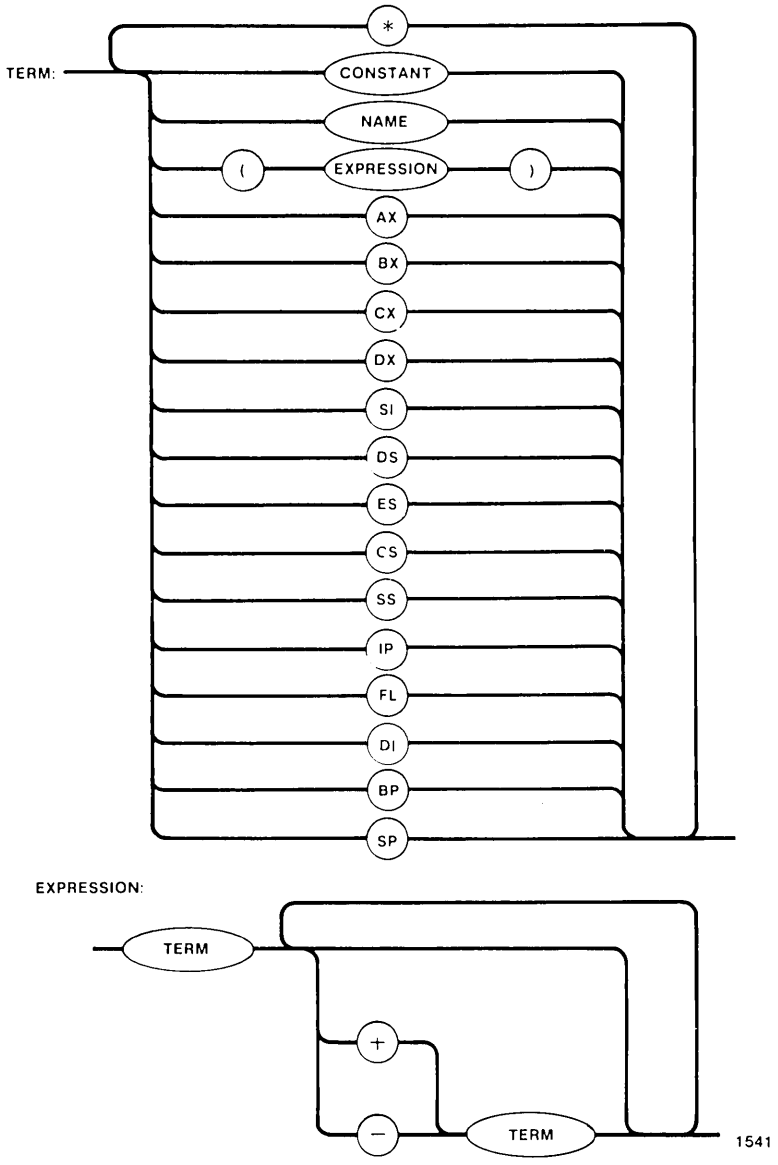


Figure 3-2. Syntax Diagrams for Term and Expression

DEBUGGER COMMANDS

4.1 INTRODUCTION TO DEBUGGER COMMANDS

This chapter presents the details of the Debugger commands. It is divided into several sections, each of which describes a related group of commands. The command groups are as follows:

- Symbolic Name Commands
- Breakpoint Commands
- Memory Commands
- Commands to Inspect iRMX I Objects
- Commands to View Object Lists
- Commands to Exit the Debugger

Each section contains a general information portion followed by detailed command descriptions.

Following this introduction is a command directory. This directory, which lists the commands in alphabetical order, includes short descriptions and page numbers for the complete descriptions. Non-alphabetic commands are listed at the end of the dictionary.

The first occurrence of each command is printed in blue ink and appears on the outside upper corner of the page; subsequent occurrences are printed in black ink. In the examples, boldface type (**this is boldface type**) is used to indicate an entry you make from your terminal.

Because the iRMX I operating system can run under several microprocessors, the generic term "CPU" will be used instead of 8086, 80186, 8088, 80188, 80286 and 386™.

4.2 COMMAND DIRECTORY

4.3 SYMBOLIC NAME COMMANDS

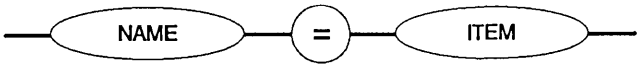
For your convenience during debugging, the Debugger supports the use of alphanumeric variable names that stand for numerical quantities. The Debugger accesses the names and their values from any of the following sources:

- A Debugger-maintained symbol table. The table contains name/value pairs catalogued by the Debugger as numeric variables. This section describes commands for defining, changing, listing, and deleting numeric variables.
- The object directory of the current job. The current job is defined as the job that contains the breakpoint task. (The command that establishes the breakpoint task is in the "Breakpoint Commands" section of this chapter.) If no breakpoint task exists, the current job is the root job.
- The object directory of the root job.

When you use a symbolic name that is not the name of a breakpoint variable, the Debugger searches these sources in the order just listed.

Suppose that you want to refer to a particular task using .TASK001. If the task is catalogued in the object directory of either the root job or the current job, then the Debugger will go to the appropriate directory and fetch a token for the task whenever the name .TASK001 is used in a Debugger command. If the task is not so catalogued, you can use VJ (view job), IJ (inspect job), VT (view task), or IT (inspect task) to deduce a token for the task. Then you can define .TASK001 to be a numeric variable whose value is that token.

This command changes the value of an existing numeric variable. The syntax for this command is as follows:



W-1058

Parameters

NAME	Name of an existing numeric variable.
ITEM	An expression or the name of a CPU segment register. The value of ITEM is associated with the variable name NAME.

Description

This command removes from the Debugger symbol table the value originally associated with NAME and replaces it with the value of ITEM.

Examples

```
.TASKA = 2F00
*
```

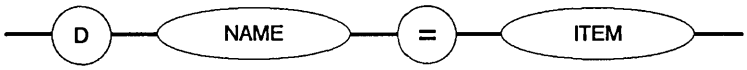
This command changes the value of .TASKA to 2F00h.

```
.TASKA = .TASKB
*
```

This command changes the value of .TASKA to that of .TASKB.

DEFINING NUMERIC VARIABLES -- D

This command associates a variable name with a numeric value. The syntax for the D command is as follows:



W-1059

Parameters

NAME	Name of the variable. This must be a period followed by up to 11 characters, the first of which must be alphabetic. The other characters can be alphabetic, numeric, question marks (?), or dollar signs ($).
ITEM	An expression or the name of a CPU segment register. The value of ITEM is associated with the variable name NAME.

Description

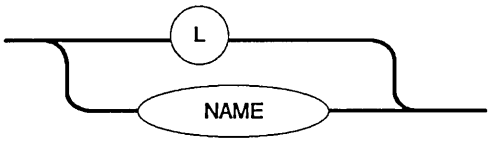
This command places NAME and the value of ITEM into the Debugger symbol table. You can use this command to create symbolic names for tokens, registers, or any other values. Then, you can use the symbolic names in other Debugger commands instead of entering the actual hexadecimal values.

Examples

```
D .TASKA = 2DC3
*
```

This command creates a symbol called .TASKA in the Debugger's local symbol table and assigns this symbol the value 2DC3h.

This command lists numeric and breakpoint variable names and their associated values. The syntax for the L command is as follows:



W-1073

Parameter

NAME	Name of an existing numeric or breakpoint variable. If entered, the Debugger lists the name and value of the indicated name only.

Description

The L command lists all numeric and breakpoint variable names and their associated values. (Breakpoint variables are described in the "Breakpoint Commands" section of this chapter.) Specifying NAME instead of L causes only one pair to be listed. In either case, one pair is listed per line in this format:

```
NAME=xxxx
```

where xxxx is the associated value.

Examples

```
L
BP=2DC3:00FF
MBOX            2F34
TASKA           2DC3
TASKB           2B8C
TASKC           2D8A
TASKD           2CEF
*
```

LISTING NUMERIC AND BREAKPOINT VARIABLES -- L

This command lists the names and values of all the numeric and breakpoint variables in the Debugger's local symbol table. It lists one breakpoint variable (.BP) and five numeric variables (.MBOX, .TASKA, .TASKB, .TASKC, and .TASKD).

```
.TASKA
TASKA=2DC3
*
```

This command lists the value associated with the variable .TASKA.

This command deletes a numeric variable. The syntax for the Z command is as follows:

W-1060

Parameter

NAME	Name of an existing numeric variable to be deleted.

Description

This command removes the NAME and associated value from the Debugger's symbol table.

Example

```
Z .TASKA
*
```

This command deletes the numeric variable .TASKA.

4.4 BREAKPOINT COMMANDS

The Debugger enables you to set, change, view, or delete breakpoints. You set a breakpoint by defining an act that a task can perform. When a task performs the act, it incurs the breakpoint, causing its execution to cease. The Debugger supports three kinds of breakpoints:

- Execution breakpoint. A task incurs an execution breakpoint when it executes an instruction at a designated location in memory.
- Exchange breakpoint. A task incurs an exchange breakpoint when it performs a designated type of operation (send or receive) at a designated exchange.
- Exception breakpoint. A task incurs an exception breakpoint if its exception handler has been declared to be the Debugger and the task causes an exceptional condition of the type that invokes its exception handler.

When a task incurs a breakpoint (of any type), three events occur automatically:

- The task is placed in a pseudostate called "broken". Depending on the breakpoint options selected, the broken task and the tasks in the containing job might be suspended.
- If suspended, the broken task (and suspended tasks, if any) is placed on a Debugger-maintained list called the breakpoint list. You can resume a task on the breakpoint list or you can remove it from the list.
- At the terminal, a display informs you that a breakpoint has been incurred. It also provides information about the event.

Each task on the breakpoint list is assigned a breakpoint state, which reflects the kind of breakpoint last incurred by the task. The states are as follows:

X ---	The task incurred an execution breakpoint.
E ---	The task incurred an exchange breakpoint.
Z ---	The task incurred an exception breakpoint.
N ---	The task was placed on the breakpoint list when another task in the same job incurred a breakpoint which had been set with the DB command (described later) using the J option.

You set an execution or exchange breakpoint with the DB command by defining a breakpoint variable and assigning it a breakpoint request. The request specifies to the Debugger the nature of the breakpoint, and the variable provides you with a convenient means of talking to the Debugger about the breakpoint. Using the breakpoint variable, you can cancel the breakpoint or replace it with a new one.

If you want to monitor a particular task that has not necessarily incurred a breakpoint, you can designate it to be the breakpoint task. If the task is not on the breakpoint list when you do this, the task is suspended and is not placed on the breakpoint list. After designating a breakpoint task, you can examine and alter some of its registers. You can also ascertain the breakpoint state of the task. When ready, you can easily resume the task.

The Debugger displays information when a task incurs a breakpoint. The format of the display depends on the kind of breakpoint incurred.

When the task is accessing a region, the Debugger cannot process breakpoints normally. When this situation occurs, the Debugger displays the following message:

```
TASK IN REGION INCURRED BREAKPOINT:  bp-var, TASK=jjjjJ/ttttT
      FULL BREAKPOINT INFORMATION NOT AVAILABLE
      TASK NOT PLACED ON BREAKPOINT LIST
```

where:

bp-var	The name of the breakpoint variable.
jjjj	A token for the task's job.
tttt	A token for the task.

4.4.1 Execution Breakpoint Display

The Debugger displays the following information when a task incurs an execution breakpoint:

```
bp-var:  E, TASK=jjjjJ/ttttq, CS=cccc, IP=iiii
```

where:

bp-var	The name of the breakpoint variable.
jjjj	A token for the task's job.
tttt	A token for the task.
q	Either T (for task) or * (indicating that the task has overflowed its stack).
cccc	The base of the code segment in which the breakpoint was set.
iiii	The offset of the breakpoint within its code segment.

4.4.2 Exchange Breakpoint Display

The Debugger displays the following information when a task incurs an exchange breakpoint:

```
bp-var:  a, EXCH=jjjjJ/xxxxe, TASK=jjjjJ/ttttq, ITEM=item
```

where:

bp-var	The name of the breakpoint variable.
a	Indicates which kind of operation (S for send or R for receive) caused the breakpoint to be incurred.
jjjj	A token for the job containing the exchange whose token follows.
xxxx	A token for the exchange.
e	Indicates the type of the exchange (M for mailbox, S for semaphore, R for region).
tttt	A token for the task.
q	Either T (for task) or * (indicating that the task has overflowed its stack).
item	One of the following: • If the exchange is a mailbox, this field lists a pair of tokens, in this form: jjjjJ/oooot, where: jjjj — A token for the mailbox's containing job. oooo — A token for the object being sent or received. t — The type of the object being sent or received (J for job, T for task, M for mailbox, S for semaphore, G for segment, R for region, X for extension, C for composite). • If the type of operation was receive, but no object was there to be received, item is 0000. • If the exchange is a semaphore, this field lists the number of units held by the exchange.

4.4.3 Exception Breakpoint

The Debugger displays the following information when a task incurs an exception breakpoint:

```
EXCEPTION:  jjjjJ/ttttT, CS=cccc, IP=iiii, TYPE=wwww, PARAM=vvvv
```

where:

jjjj	A token for the job containing the task that caused the exception condition.
tttt	A token for the task that caused the exception condition.
cccc and iiii	Respectively, the contents of the CS and IP registers when the exception condition occurred.
wwww	The numerical value of the exception code; reflects the nature of the exception condition. Refer to the iRMX reference manuals for the mnemonic condition codes and their numerical equivalents.
vvvv	The number (0001 for first, 0002 for second, etc.) of the parameter that caused the exception condition. If no parameter was at fault, vvvv is 0000.

4.4.4 Exception Breakpoint Differences

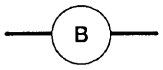
Exception breakpoints differ from execution and exchange breakpoints in several respects:

- You cannot set, change, view, or delete exception breakpoints by using the Debugger commands. Instead, each task can set an exception breakpoint by declaring the Debugger to be its exception handler. The task can then delete the breakpoint by declaring a different exception handler. However, like the other kinds of breakpoints, once a task incurs an exception breakpoint and is placed on the breakpoint list, you can cause it to resume execution with the same command (the G command) used to resume other tasks on the breakpoint list.exception handler
- You set exception breakpoint for a particular task, while you set execution and exchange breakpoints for no particular task; any task can incur such a breakpoint.execution breakpoints
- The Debugger does not know an exception breakpoint by a breakpoint variable name.

Exception breakpoints are handled different from execution and exchange breakpoints. For example, exception breakpoints cannot be viewed, but the other breakpoints can. Wherever this distinction applies, this chapter points it out.

VIEWING BREAKPOINT PARAMETERS -- B

This command displays the breakpoint parameters. The syntax for the B command is as follows:



W-1061

Description

The B command performs these three functions:

- Displays the breakpoint list
- Displays the breakpoint task
- Displays the breakpoint variables

Breakpoint List Display

The B command first displays the breakpoint list in the following format:

```
BL=jjjjJ/ttttT(s) jjjjJ/ttttT(s) ... jjjjJ/ttttT(s)
```

where:

jjjj	A token for the job containing the task whose token follows.
tttt	A token for a task on the breakpoint list.
s	The breakpoint state of a task. Possible values are X (for execution), E (for exchange), Z (for exception), and N (for null).

Breakpoint Task Display

The second effect of the B command is to display the breakpoint task originally selected with the BT command. The format of this display is as follows:

```
BT=jjjjJ/ttttT(s)
```

where:

jjjj	A token for the job containing the breakpoint task.
tttt	A token for the breakpoint task.
s	The breakpoint state of the breakpoint task. Possible values are X (for execute), E (for exchange), Z (for exception), and N (for null).

If there is no breakpoint task, the display is

```
BT=0
```

Breakpoint Variables Display

Finally, the B command displays the breakpoint variables. The format of the display depends on whether the variables are execution or exchange variables.

Execution breakpoints are displayed as:

```
bp-var = xxxx:yyyy z ops
```

where:

bp-var	The name of the breakpoint variable.
xxxx	The base portion of the address at which the breakpoint is set.
yyyy	The offset portion of the address at which the breakpoint is set.
z	Indicates whether a task (T) or all the tasks in a job (J) are to be suspended and placed on the breakpoint list when the breakpoint is incurred.
ops	Indicates the breakpoint options. If any are present, they can be C (for Continue task) and/or D (for Delete breakpoint).

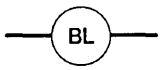
Exchange breakpoints are displayed as:

```
bp-var = xxxx a z ops
```

where:

bp-var	The name of the breakpoint variable.
xxxx	A token for the exchange at which the breakpoint is set.
a	Indicates the kind of breakpoint activity at the exchange, either S (for Send), R (for Receive), or SR (for both).
z	Indicates whether a task (T) or all the tasks in a job (J) are to be suspended and placed on the breakpoint list when the breakpoint is incurred.
ops	Indicates the breakpoint options. If any are present, they can be C (for Continue task) and/or D (for Delete breakpoint).

This command displays the breakpoint list. The syntax for the BL command is as follows:



W-1062

Description

The BL command displays the entire breakpoint list at the terminal, as follows:

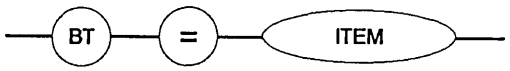
```
BL=jjjjJ/ttttT(s) jjjjJ/ttttT(s) ... jjjjJ/ttttT(s)
```

where:

jjjj	A token for the job containing the task whose token follows.
tttt	A token for a task.
s	The breakpoint state of a task. Possible values are X (for execution), E (for exchange), Z (for exception), and N (for null).

ESTABLISHING THE BREAKPOINT TASK -- BT

This command designates a task to be the breakpoint task. The syntax for the BT command is as follows:



W-1063

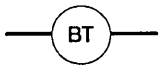
Parameter

ITEM — A token for an existing task.

Description

The task designated by ITEM becomes the breakpoint task. The Debugger suspends the task but does not place it on the breakpoint list.

This command lists the job and task tokens associated with the breakpoint task. The syntax for the BT command is as follows:



W-1064

Description

This command displays the following information about the breakpoint task:

```
BT=jjjjJ/ttttT(s)
```

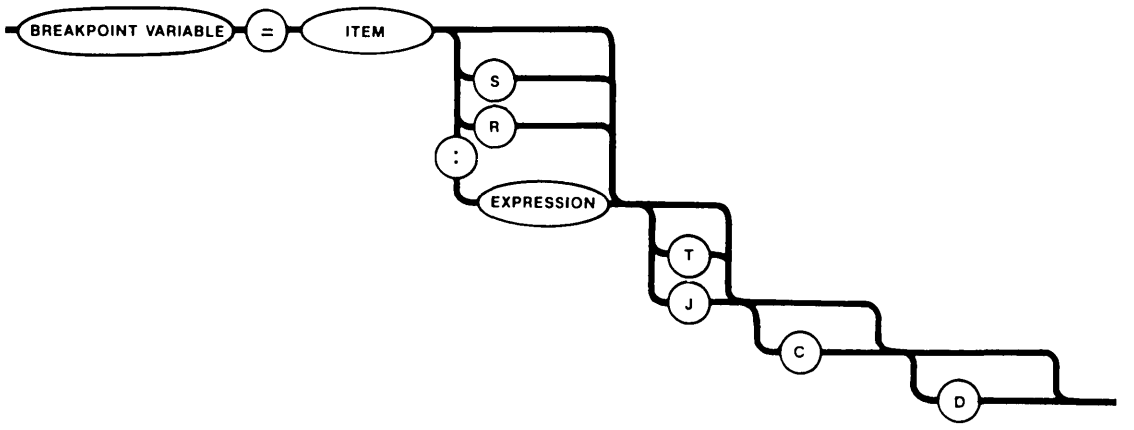
where:

jjjj	A token for the job containing the breakpoint task.
tttt	A token for the breakpoint task.
s	The breakpoint state of the breakpoint task. Possible values are X (for execute), E (for exchange), Z (for exception), and N (for null).

If there is no breakpoint task, the Debugger displays the following:

```
BT=
```

This command changes an existing breakpoint. The syntax for this command is as follows:



1551

Parameters

BREAKPOINT VARIABLE	An existing Debugger breakpoint name. If the Debugger's symbol table does not already contain this name, an error message will appear on the terminal.
ITEM and EXPRESSION	If you are changing an execution breakpoint, use ITEM with EXPRESSION to specify the address of the breakpoint. ITEM must contain the base portion of the address, followed by ":" and an EXPRESSION, which must contain the offset portion. If you are changing an exchange breakpoint, ITEM must contain a token for an exchange.
S and R	To be used only when changing an exchange breakpoint. S means that the exchange breakpoint is for senders only, while R designates receivers only. If you want to set an exchange breakpoint for both senders and receivers, omit both S and R, as well as both ":" and EXPRESSION.
T and J	Indicate which tasks are to be put on the breakpoint list when a breakpoint is incurred. T indicates only the task that incurred the breakpoint, while J indicates all of the tasks in that task's job. If neither T nor J is present, T is assumed.

C — Continue task execution option. This option directs the Debugger not to "break" tasks that incur the breakpoint, and not to put them on the breakpoint list. When a task incurs such a breakpoint, the Debugger generates a breakpoint display, but the task continues to run.

D — Delete breakpoint option. This option directs the Debugger to delete the breakpoint after it is first incurred by a task. The Debugger generates a breakpoint display and, unless the C option is also specified, places the task that incurred the breakpoint on the breakpoint list.

Description

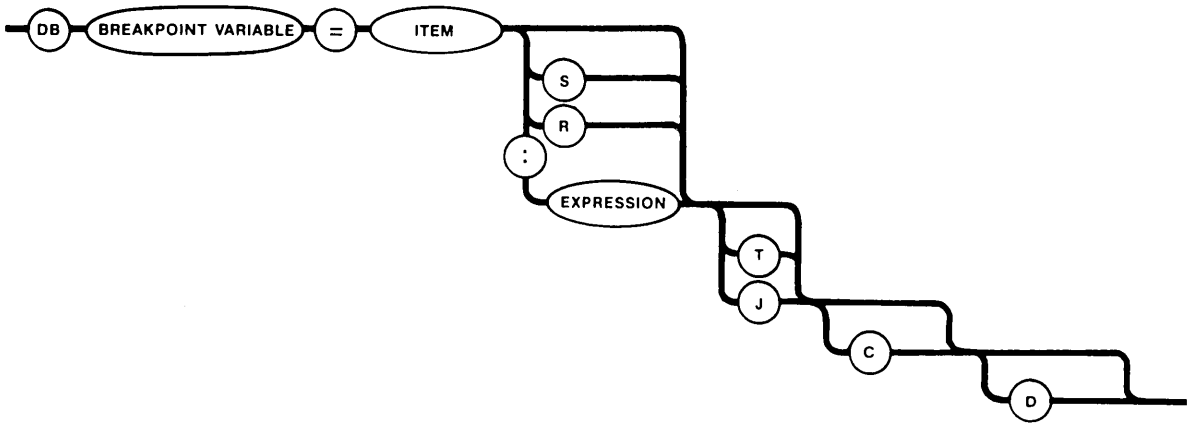
This command deletes the breakpoint associated with the breakpoint variable name and replaces it with a new breakpoint, as specified in the command. The breakpoint variable name can be used when deleting or changing the breakpoint.

Example

```
.BPOINT
BPOINT=2F34 S T C
*
.BPOINT = 2D2A S C
*
.BPOINT
BPOINT=2D2A S C
*
```

In this example, the user lists a breakpoint variable, changes it, and lists it again.

This command defines an execution or exchange breakpoint. The syntax for the DB command is as follows:



Parameters

BREAKPOINT VARIABLE	A Debugger name that identifies the breakpoint. This name must consist of a period followed by up to 11 characters, the first of which must be alphabetic. The other characters can be alphabetic, numeric, question marks (?), or dollar signs ($). If the Debugger's symbol table already contains this name, an error message will appear on the terminal.
ITEM	If you are setting an execution breakpoint, use ITEM with EXPRESSION to specify the address of the breakpoint. ITEM must contain the base portion of the address, followed by ":" and an EXPRESSION, which must contain the offset portion. If you are setting an exchange breakpoint, ITEM must contain a token for an exchange.
S and R	To be used only when setting an exchange breakpoint. S means that the exchange breakpoint is for senders only, while R indicates receivers only. If you want to set an exchange breakpoint for both senders and receivers, omit both S and R, as well as both ":" and EXPRESSION.
EXPRESSION	Use only when setting an execution breakpoint. EXPRESSION must contain the offset portion of the address of the execution breakpoint.

T and J	Indicates which tasks are to be put on the breakpoint list when a breakpoint is incurred. T indicates only the task that incurred the breakpoint, while J indicates all of the tasks in that task's job. The default is T.
C	Continues task execution option. This option directs the Debugger not to "break" tasks that incur the breakpoint, and not to put them on the breakpoint list. When a task incurs such a breakpoint, the Debugger generates a breakpoint display, but the task continues to run.
D	Deletes breakpoint option. This option directs the Debugger to delete the breakpoint after it is first incurred by a task. The Debugger generates a breakpoint display and, unless the C option is also specified, places the task that incurred the breakpoint on the breakpoint list.

Description

The DB command sets a breakpoint of the type indicated in the remainder of the command line. The name designated as the breakpoint variable can be used when altering or deleting the breakpoint.

Examples

```
DB .BP = 2DC3:0FF
*
```

This command defines an execution breakpoint at address 2DC3:0FF and assigns the name .BP to this breakpoint. When a task incurs this breakpoint, only the task itself is placed on the breakpoint list.

```
DB .BPOINT = .MBOX S C
*
```

This command defines an exchange breakpoint at the mailbox whose token is specified by the numeric variable .MBOX.

This command displays information about a particular breakpoint. The syntax for this command is as follows:

W-1065

Parameter

BREAKPOINT
VARIABLE — The name of an existing breakpoint to be examined.

Description

The Debugger displays two kinds of output, depending on whether the specified breakpoint variable represents an execution or an exchange breakpoint. Exception breakpoints cannot be examined.

Execution Breakpoint Output

If the designated breakpoint is an execution breakpoint, the Debugger sends the following display to the terminal:

```
bp-var=xxxx:yyyy  z ops
```

where:

bp-var	The name of the breakpoint variable.
xxxx	Base portion of the breakpoint's address.
yyyy	Offset portion of the breakpoint's address.
z	Indicates whether a single task (T) is to be "broken" and placed on the breakpoint list or all tasks in a job (J) are to be suspended and placed on the breakpoint list, when the breakpoint is incurred.
ops	Indicates the breakpoint options. If any are present, they can be C (for Continue task) and/or D (for Delete breakpoint).

EXCHANGE BREAKPOINT OUTPUT

If the designated breakpoint is an exchange breakpoint, the Debugger sends the following display to the terminal:

```
bp-var=xxxx a z ops
```

where:

bp-var	The name of the breakpoint variable.
xxxx	A token for the exchange at which the breakpoint is set.
a	Indicates the kind of breakpoint activity at the exchange, either S (for send), R (for receive), or SR (for both).
z	Indicates whether a single task (T) is to be "broken" and placed on the breakpoint list or all tasks in a job (J) are to be suspended and placed on the breakpoint list, when the breakpoint is incurred.
ops	Indicates the breakpoint options. If any are present, they can be C (for Continue task) and/or D (for Delete breakpoint).

Examples

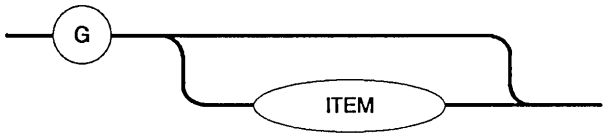
```
.BP
BP=2DC3:00FF T
*
```

This command lists the address of the execution breakpoint associated with the variable .BP. It also indicates that the task is to be "broken" only if a breakpoint is encountered.

```
.BPOINT
BPOINT=2F34 S T C
*
```

This command lists the address of the exchange breakpoint associated with the variable .BPOINT. The S, T, and C indicate that only tasks that send messages to the exchange will incur the breakpoint, only the task that incurs the breakpoint will be "broken," and the task will continue processing after incurring the breakpoint.

This command resumes execution of a task on the breakpoint list or the breakpoint task. The syntax for the G command is as follows:



W-1074

Parameter

ITEM — A token for a task on the breakpoint list or the breakpoint task. If the token is not for a task on the breakpoint list or is not the breakpoint task, an error message is displayed. If this parameter is omitted, the breakpoint task is assumed.

Description

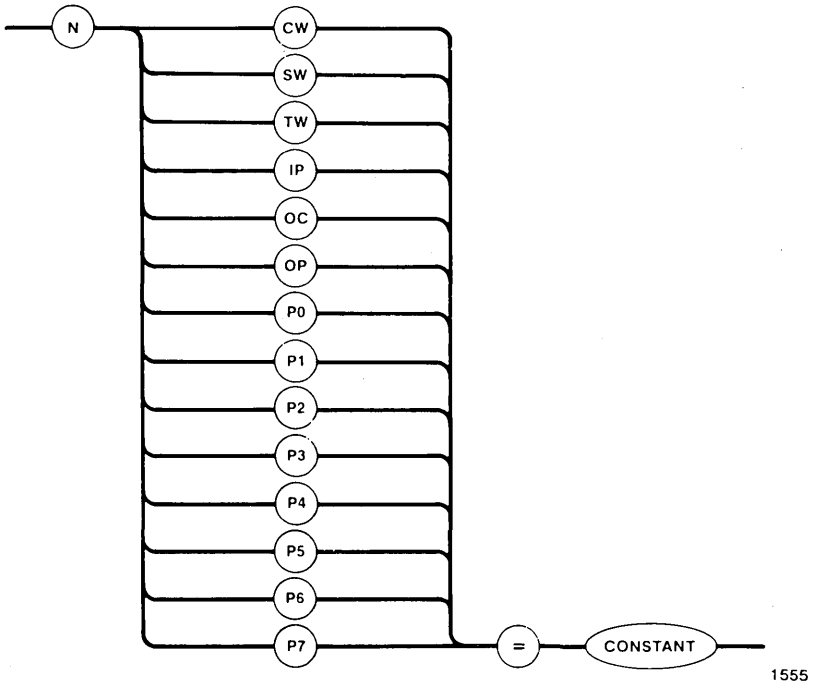
The G command applies to the breakpoint task if ITEM is not present. Otherwise, it applies to the task on the breakpoint list whose token is represented by ITEM.

The G command resumes execution of the designated task. If the task is in the broken state, it is made ready. If in the suspended state, its suspension depth is decreased by one.

If the G command is invoked without ITEM when there is no breakpoint task, an error message is displayed.

ALTERING THE BREAKPOINT TASK'S NPX REGISTERS -- N

This command modifies the breakpoint task's Numeric Processor Extension (NPX) register values. This command applies only to tasks specified at creation as having the ability to use the NPX. The syntax for the N command is as follows:



Parameters

CW, SW, TW, IP, OC, OP, P0 through P7 — Names of the breakpoint task's NPX registers, as follows:

Name	Description
CW	Control Word
SW	Status Word
TW	Tag Word
IP	Instruction Pointer
OC	Operation Code
OP	Operand Pointer
P0-P7	Stack elements

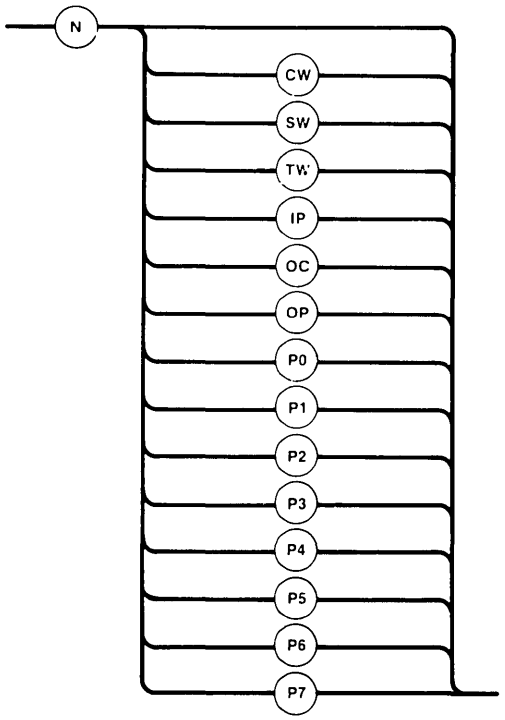
CONSTANT — A hexadecimal number used for the new register value. CONSTANT can specify an 80-bit value for registers P0 through P7, a 20-bit value for registers IP and OP, and a 16-bit value for the remaining registers. If this value is too large for the specified register, the Debugger displays a SYNTAX ERROR message.

Description

This command requests that the breakpoint task's NPX register, as specified in the command request, be updated with the value of CONSTANT. This command applies only to tasks specified at creation as using the NPX.

VIEWING THE BREAKPOINT TASK'S NPX REGISTERS -- N

This command displays the breakpoint task's Numeric Processor Extension (NPX) register values. This command applies only to tasks specified at creation as having the ability to use the NPX. The syntax for this command is as follows:



Parameters

CW, SW, TW, IP,
OC, OP, P0
through P7

Names of the breakpoint task's NPX registers, as follows:

Name	Description
CW	Control Word
SW	Status Word
TW	Tag Word
IP	Instruction Pointer
OC	Operation Code
OP	Operand Pointer
P0-P7	Stack elements

If no name is specified, the Debugger displays values for all registers.

Description

This command lists NPX register values for the breakpoint task. It applies only to tasks specified at creation as using the NPX. If the command is simply "N," then all of the breakpoint task's NPX registers are displayed, in the following format:

```
NCW = xxxx          NSW = xxxx          NTW = xxxx
NIP = xxxxx         NOC = xxx           NOP = xxxxx
NP0 = xxxxxxxxxxxxxxxxxxx
NP1 = xxxxxxxxxxxxxxxxxxx
NP2 = xxxxxxxxxxxxxxxxxxx
NP3 = xxxxxxxxxxxxxxxxxxx
NP4 = xxxxxxxxxxxxxxxxxxx
NP5 = xxxxxxxxxxxxxxxxxxx
NP6 = xxxxxxxxxxxxxxxxxxx
NP7 = xxxxxxxxxxxxxxxxxxx
NES = xxxx
```

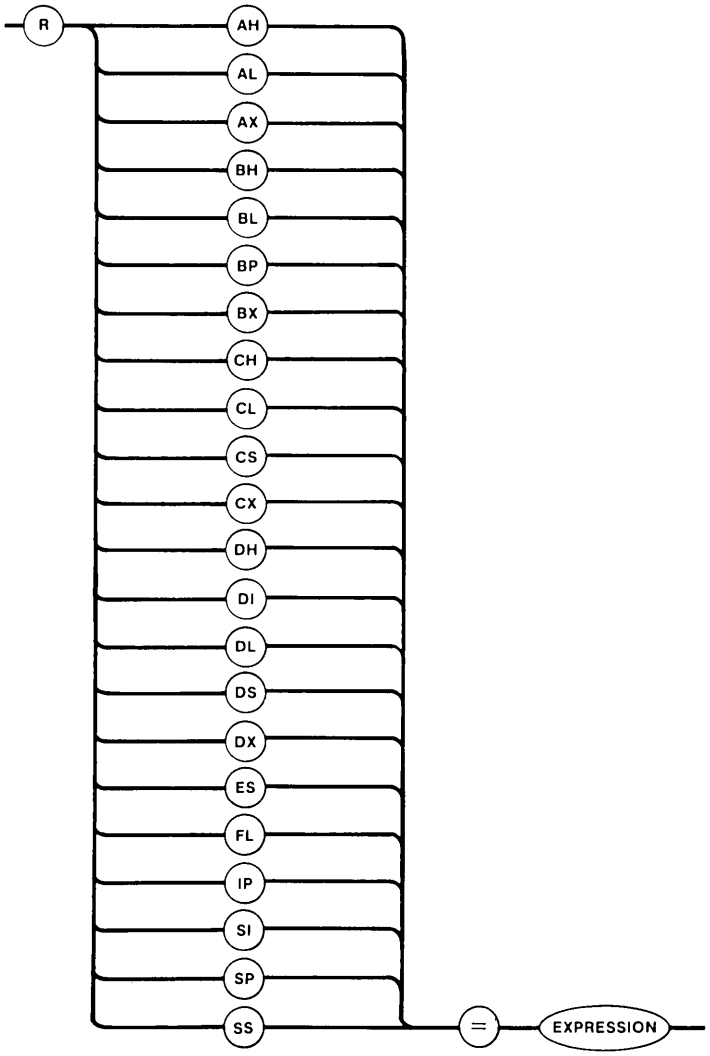
The size of the field indicates the number of hexadecimal digits that the Debugger displays.

Registers P0 through P7 are 80-bit registers that the Debugger displays in temporary real format.

The NES field contains the value of the NPX Status Word if an NPX exception caused the breakpoint task to be broken. The value for this field, under all other circumstances, is NONE.

If the breakpoint task does not use the NPX, the Debugger returns an error message.

This command alters one of the breakpoint task's CPU register values. The syntax for the R command is as follows:



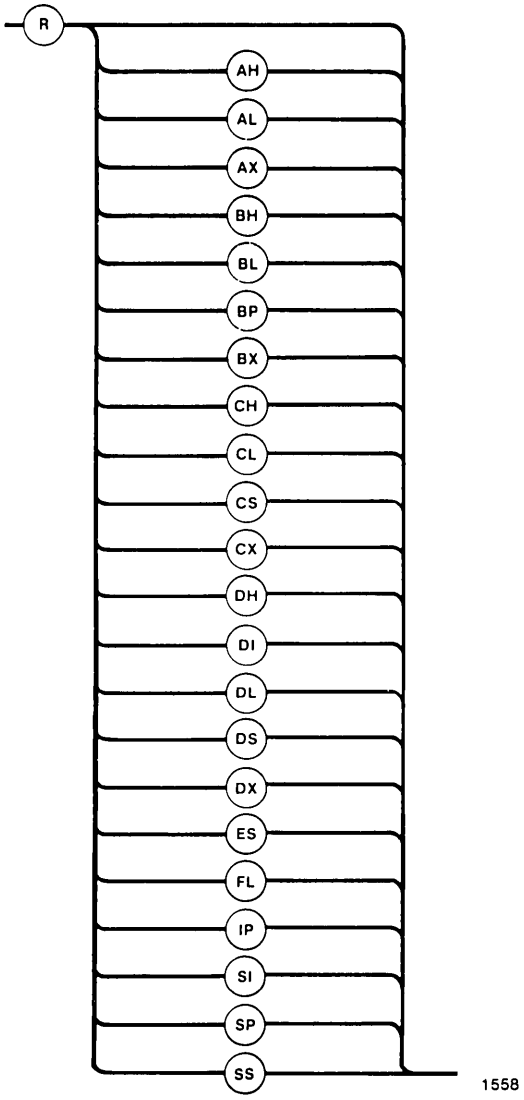
Parameters

AH, AL, AX, BH, BL, BP, BX, CH, CL, CS, CX, DH, DI, DL, DS, DX, ES, FL, IP, SI, SP, SS	Names of the breakpoint task's CPU registers.
EXPRESSION	A Debugger expression whose value is used for the new register value. If this value is too large to fit in the designated register, the Debugger fills the register with the low-order bytes of the value.

Description

This command requests that the breakpoint task's register, as specified in the command request, be updated with the value of the EXPRESSION. However, if the breakpoint task is in the null breakpoint state, its register values cannot be altered by the R command.

This command lists one or all of the breakpoint task's CPU registers. The syntax for the R command is as follows:



Parameters

AH, AL, AX, BH, BL, BP, BX, CH, CL, CS, CX, DH, DI, DL, DS, DX, ES, FL, IP, SI, SP, SS	Names of the breakpoint task's CPU registers. If no name is specified, the Debugger displays values for all registers.

Description

This command lists CPU register values for the breakpoint task. If the command is simply "R," then all of the breakpoint task's registers are displayed, in the following format:

```
RAX=xxxx    RSI=xxxx    RCS=xxxx    RIP=xxxx
RBX=xxxx    RDI=xxxx    RDS=xxxx    RFL=xxxx
RCX=xxxx    RBP=xxxx    RSS=xxxx
RDX=xxxx    RSP=xxxx    RES=xxxx
```

If the command has the form Ryy, where yy is the register name, then the contents of the specified register are displayed, either as

```
Ryy=xxxx
```

or as

```
Ryy=xx
```

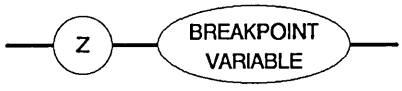
depending on whether yy is a byte-size register (like AH) or a word-size register (like AX).

If the breakpoint task is in the null breakpoint state, only its BP, SP, CS, DS, SS, IP, and FL register contents are displayed. The remaining register displays consist of question marks.

In certain circumstances the breakpoint task, when suspended, is in a state that prevents the Debugger from obtaining its register contents. If this is the case, the Debugger displays question marks for all registers.

DELETING A BREAKPOINT -- Z

This command deletes a breakpoint. The syntax for the Z command is as follows:



W-1066

Parameter

BREAKPOINT VARIABLE	Name of an existing Debugger breakpoint to be deleted.

Description

The Z command deletes the specified breakpoint and removes the breakpoint variable name from the Debugger's symbol table.

Example

```
Z .BP
*
```

This command deletes the breakpoint associated with the variable .BP and removes .BP from the Debugger's symbol table.

4.5 MEMORY COMMANDS

The commands in this section enable you to inspect or modify the contents of absolute memory locations. Figure 4-1 illustrates the syntax for all commands in this section.

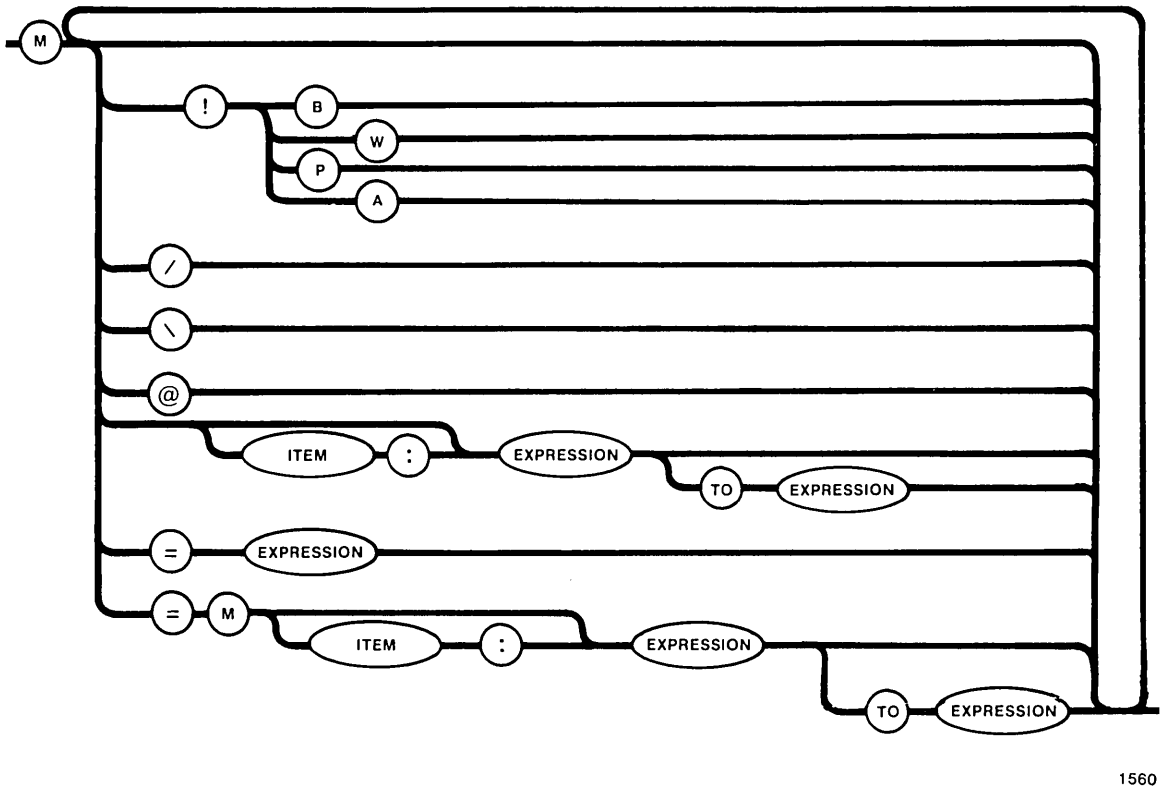


Figure 4-1. Syntax Diagram for Memory Commands

As Figure 4-1 illustrates, all memory commands begin with "M." A variety of parameters can be specified with "M"; these parameters are grouped into the following basic options:

- Setting current display mode. This option begins with "!."
- Changing memory locations. This option includes "=."

- Displaying memory locations. This option consists of the remaining parameters.

This section discusses these three groups as separate commands; however, you can combine any number of "M" command options in a single command, as Figure 4-1 shows.

The following command descriptions mention the current display mode, the current segment base, the current offset, the current address, and the display of memory locations, defined as follows:

- The **current display mode** determines how memory values are interpreted for display. The possible modes are designated by B (byte), W (word), P (pointer), and A (ASCII). The effects of these modes can easily be understood by an example.

  Suppose that memory locations 042B through 042E contain, respectively, the values 25, F3, 67, and 4C. If you ask for the display of the memory at location 042B, then the effects, which depend on the current display mode, are as follows:

Current Display Mode	Display
B	25
W	F325
P	4C67:F325
A	%

  Observe that words and pointers are displayed from high-order (high address) to low-order (low address).

- If a location contains a value that does not represent a printable ASCII character, and the current display mode is A, then the Debugger prints a period. The initial current display mode is B.
- The value of the **current segment base** is always the value of the most recently used CPU segment base. The initial value of the current segment base is 0.
- The **current offset** is a value the Debugger maintains and uses when a memory location is referenced with no offset value. Except when the current offset has been modified by certain options of the M command, the current offset is always the value of the most recently used offset. The initial value of the current offset is 0.
- The **current address** is the memory address computed from the combination of the current segment base and the current offset.
- When memory locations are displayed, the format is as follows:

```
xxxx:yyyy=value
```

- where xxxx and yyyy are the current segment base and current offset, respectively, and value is a byte, word, pointer, or ASCII character, depending on the current display mode. If several contiguous memory locations are being requested at one time, each line of display is as follows:

```
xxxx:yyyy=value value value ... value
```

where xxxx, yyyy, and value are as previously described, and xxxx:yyyy represents the address of the first value on that line.

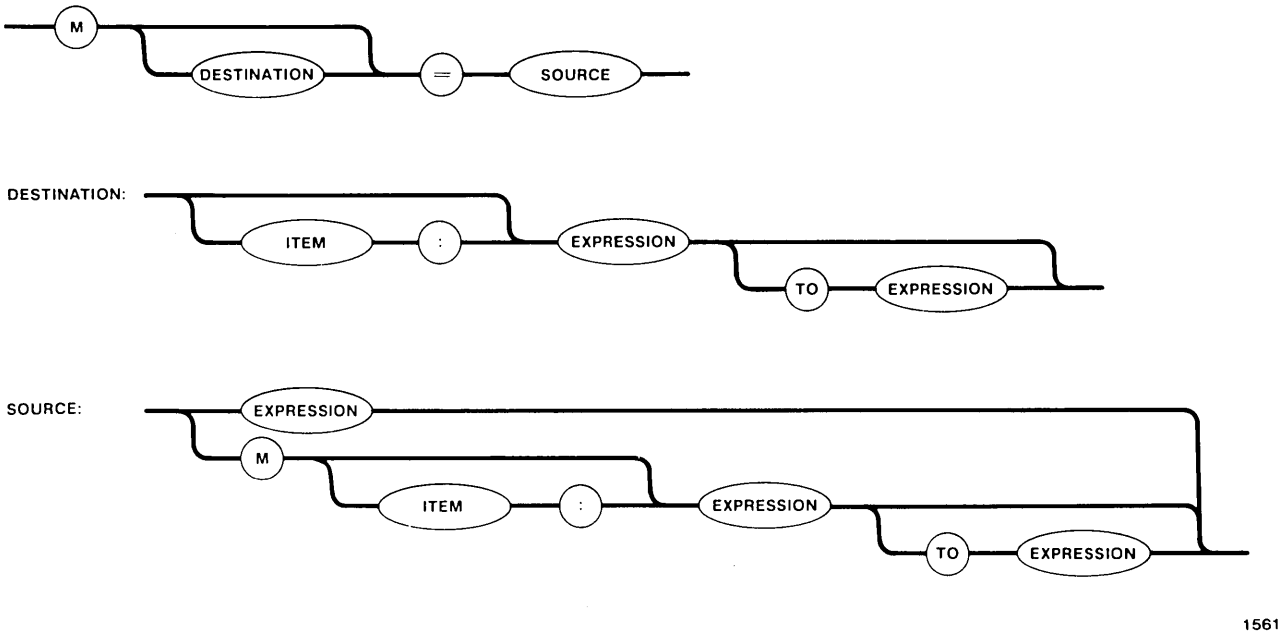
The first such line begins with the first address in the request and continues to the end of that (16-byte) paragraph. If additional lines are required to satisfy the request, each begins at an offset that is a multiple of 16 (10H).

This command changes the contents of designated RAM locations.

CAUTION

Because the Debugger is most often used during system development while your tasks, the Nucleus, the Debugger, and possibly other iRMX I components are in RAM, you should use these M command options with extreme care.

The syntax for this command is as follows:



Parameters

As shown in the syntax diagram, the parameters for this command are divided into DESTINATION and SOURCE parameters separated with an equal sign.

Destination Parameters

These parameters define the memory location or locations to be changed. All parameters change the current offset, and some of them change the current base. The valid parameter combinations are as follows:

EXPRESSION	This form of the DESTINATION option implies that the address to be changed has the current base as its base value and the value of EXPRESSION as its offset.
ITEM: EXPRESSION	This form of the DESTINATION option implies that the address to be changed has the value of ITEM as its base value and the value of EXPRESSION as its offset.
EXPRESSION TO EXPRESSION	This form of the DESTINATION option implies that a series of consecutive locations will be changed. The EXPRESSIONs determine the beginning and ending offsets, respectively. The current base is used as a base value. After memory has been changed, the current offset is set to the value of the second EXPRESSION.
ITEM: EXPRESSION TO EXPRESSION	This form of the DESTINATION option is the same as the previous one, except that ITEM is used as the base value of the locations.

If no DESTINATION option is specified, the location specified by the current segment base and current offset is changed. However, if the previous command was a "Display Memory" command of the form

```
M EXPRESSION TO EXPRESSION
```

the entire range of locations specified in that command is changed.

Source Parameters

These parameters define the information to be placed into the DESTINATION memory. The valid parameter combinations are as follows:

EXPRESSION	This form of the SOURCE option can be used only if the current display mode is byte or word. It implies that the value represented by EXPRESSION will be copied into the byte or word at the current address. However, if the DESTINATION option (supplied or default) specified a range of locations, this option instead copies the value of EXPRESSION into each byte or word in DESTINATION.

Examples:

1. When the DESTINATION option did not specify a range of values:

```
M = 4C
0400:0008 09
0400:0008 4C
*
```

This example changes the contents of the current location (0400:0008) from 09 to 4C. Notice that the Debugger displays both the old and the new contents of memory.

2. When the DESTINATION option specified a range of values:

```
M 1 TO 4
0400:0001 06 07 08 09
*
M = 4C
0400:0001 06 07 08 09
0400:0001 4C 4C 4C 4C
*
```

In this example, because the previous command was an examination of a range of memory, the command to change memory changes the entire range of memory.

M EXPRESSION

This form of the SOURCE option uses the current segment base and the offset indicated by the value of EXPRESSION to compute an address. It copies the value at that computed address into the location specified by the current address.

However, if the DESTINATION option (supplied or default) specified a range of locations, the value at the computed address is instead copied to each location in the destination field.

Examples:

1. When the DESTINATION option did not specify a range of values:

```
M 9
0400:0009 11
*
M = M 6
0400:0009 11
0400:0009 4C
*
```

This example replaces the value in location 4000:0009 (11) with the value in location 4000:0006 (4C).

2. When the DESTINATION option specified a range of values:

```
M 100
0400:0100 FF
*
M 100 TO 103 = M 6
0400:0100 FF A0 16
0400:0100 4C 4C 4C
*
```

In this example, the command to change memory included a DESTINATION option that specified a range of values. Thus the contents of location 0400:0006 (4C) are copied into each DESTINATION location.

M ITEM: EXPRESSION

This form of the SOURCE option uses ITEM and EXPRESSION as base and offset, respectively, to compute an address. It copies the value at that computed address into the location specified by the current address. However, if the DESTINATION option (supplied or default) specified a range of locations, the value at the computed address is instead copied to each location in the destination field.

Examples:

1. When the DESTINATION option did not specify a range of values:

```
M 9
0400:0009 4C
*
M = M 300:2643
0400:0009 4C
0400:0009 21
*
```

This example takes the value in location 0300:2643 (21) and copies it into the current location (0400:0009).

2. When the DESTINATION option specified a range of values:

```
M 100 TO 103 = M 300:2643
0400:0100 4C 4C 4C 22
0400:0100 21 21 21 21
*
```

This example copies the contents of location 0300:2643 (21) into each location specified in the DESTINATION option.

M EXPRESSION TO EXPRESSION

This form of the SOURCE option uses the current segment base and, in order, the offsets indicated by the EXPRESSIONs, to compute a beginning address and an ending address. It copies the sequence of values bounded by the computed addresses to the sequence of locations that begin at the current address. However, if the DESTINATION option (supplied or default) specified a range of locations, the sequence of values bounded by the computed addresses is copied to the destination field, with the source values being truncated or repeated as required.

Examples:

1. When the DESTINATION option did not specify a range of values:

```
M 400:104
0400:0104 E1
*
M = M A TO C
0400:0104 E1 F2 0A
0400:0104 0B 0C 0D
*
```

In this example, the contents of the range of locations specified in the SOURCE option (0400:000A - 0400:000C) are copied into the range of locations that begin with the current address (0400:0104).

2. When the destination option specified a range of values:

```
M 1 TO 4 = M A TO C
0400:0001 4C 4C 4C 4C
0400:0001 0B 0C 0D 0B  (first value
*                repeated)
```

This example copies the contents of three locations (0400:000A - 0400:000C) into four locations (0400:0001 - 0400:0004). Notice that the values start repeating; 0400:0001 contains the same value as 0400:0004 (0B).

M ITEM: EXPRESSION TO EXPRESSION

This form of the SOURCE option uses ITEM as a base and the EXPRESSIONs as offsets to compute a beginning and an ending address. The sequence of values bounded by the computed addresses is copied to the sequence of locations beginning at the current address. However, if the DESTINATION option (supplied or default) specified a range of values, the sequence of values bounded by the computed addresses is copied to the destination field, with the source values being truncated or repeated as required.

Examples:

1. When the DESTINATION option did not specify a range of values:

```
D .VALUE = 2643
*
M 1
0400:0001 0B
*
M = M 300:.VALUE TO .VALUE + 4
0400:0001 0B 0C 0D 0B 4C
0400:0001 21 47 E2 C8 31
*
```

In this example, the contents of the range of locations specified in the SOURCE option (0300:2643 - 0300:2647) are copied into the range of locations that begin with the current address (0400:0001).

2. When the DESTINATION option specified a range of values:

```
M 101 TO 104
0400:0101 21 21 21 0B
*
M = M 300:2643 TO 2647
0400:0101 21 21 21 0B
0400:0101 21 47 E2 C8  (last value
*                       truncated)
```

This example copies the contents of five locations (0300:2643 - 0300:2647) into four locations (0400:0101 - 0400:0104). Notice that the value of the fifth location (0300:2647) is not copied.

Description

This command changes the contents of designated RAM locations. The DESTINATION options affect the current segment base and offset values. The SOURCE options do not affect these values.

When executing this command, the Debugger displays the contents of the designated locations, then updates the contents, and finally displays the new contents. Thus, if you inadvertently destroy some important data, you can easily access the information needed to restore it.

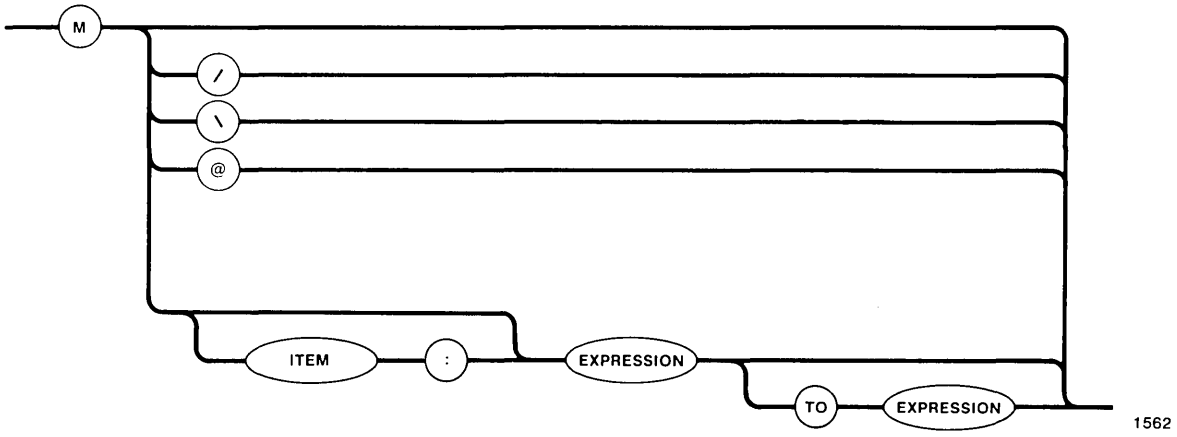
This command copies data in the byte mode. The current display mode is not affected by these copying options.

CAUTION

When using the M command, be aware of the following hazards:

- **You can modify memory within iRMX I components, such as the Nucleus and Debugger, which may jeopardize the integrity of your application system.**
- **You can request that non-RAM memory locations be modified. If you attempt to read or write to a non-RAM location, nothing happens to memory and the displays indicate that the specified locations contain zeros.**
- **A memory request might cross segment boundaries. In processing such a request, the Debugger ignores such boundaries, so don't assume that a boundary will terminate a request.**

This command displays memory locations without changing their contents. The syntax for this command is as follows:



Parameters

To avoid confusion, this section lists examples of complete commands in explaining the parameters.

M/ This option increments the current offset according to the current display mode: by one for byte or ASCII, by two for word, or by four for pointer. Then it displays the contents of the new current address.

Example:

```
M/
0400:0009 0A
*
```

This example increments the current offset and displays the address and contents of the location.

M\ This option is just like M/, except that the current offset is decremented.

Example:

```
M\
0400:0008 08
*
```

This example decrements the current offset and displays the address and contents of the location.

M — When used by itself, M is an abbreviated way of specifying M/ or M\, whichever was last used. If neither has been used in the current Debugging session, M is interpreted as an M/ request.

Example:

```
M
0400:0007 08
*
M
0400:0006 07
*
```

Since M\ was used most recently, these commands decrement the current offset before displaying the address and contents of memory.

M@ — This option sets the current offset equal to the value of the word beginning at the current address. Then the value at the adjusted current address is displayed.

Example:

```
M!B
*
M@
0400:0807 46
*
```

Even though byte mode was selected, this example sets the current offset equal to contents of the word at offset 07. From the previous example you can see that this word is indeed 0807.

M EXPRESSION — This option sets the current offset equal to the value of the EXPRESSION and displays the value at the new current address.

Example:

```
M 3
0400:0003 04
*
```

This example sets the current offset to 3 and displays the contents of that location.

M ITEM: EXPRESSION

This option is just like M EXPRESSION, except that ITEM is used as the base in the address calculation, and after the operation ITEM is the new current segment base.

Example:

```
M 300:2644
0300:2644 47
*
```

This example sets the current base to 300 and the current offset to 2644. It also displays the contents of that location.

M EXPRESSION TO EXPRESSION

This option displays the values of a series of consecutive locations. The EXPRESSIONs determine the beginning and ending offsets, respectively; the second EXPRESSION must be greater than the first. The current segment base is used as a base. After displaying the locations, the Debugger sets the current offset to the value of the second expression. If the specified range of locations is incompatible with the current display mode--for example, an odd number of locations is not compatible with the word or pointer modes--then all words or pointers that lie partially or totally inside the range are displayed.

Examples:

(1)

```
M 4 TO 6
0300:0004 15 26 37
*
```

(2)

```
M!W
    *
M 4 TO 6
0300:0004 2615 4837
*
```

These examples display a consecutive series of memory locations in both byte and word mode. Notice that the base set in the last example (300) is still used.

M ITEM: EXPRESSION TO EXPRESSION

This option is just like M EXPRESSION TO EXPRESSION, except that ITEM is used as a base in the address calculation, and after the operation ITEM is the new segment base.

Example:

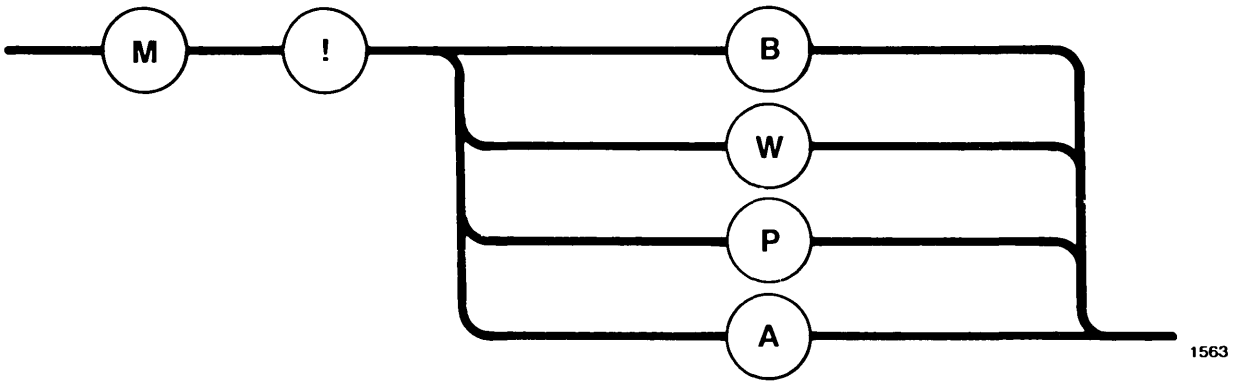
```
M!B
*
D .MEM = 100
*
M 400:.MEM TO .MEM +4
0400:0100 FF A0 16 22 E1
*
```

After setting the output mode to byte and defining a numeric variable .MEM, this example sets the base to 400 and displays five consecutive memory locations beginning with offset 100 (.MEM). At the end of the example, the current offset is 400 and the current base is 104.

Description

This command displays the contents of memory without disturbing those contents. Be aware, however, that all of the options change the current offset, and some of them change the current segment base. None changes the current display mode.

This command specifies how the Debugger will display output. The syntax for the M command is as follows:



Parameters

!	Indicates that the display mode is being changed.
B, W, P, A	Specifies the mode of display. B indicates byte mode, W indicates word mode, P indicates pointer mode, and A indicates ASCII mode.

Description

This command sets the display mode for further Debugger output. When the Debugger next displays memory, it will display the memory according to the mode specified with this command.

Examples

```
M!B
*
```

This command instructs the Debugger to display all further output in byte mode.

```
M!W
*
```

This command instructs the Debugger to display all further output in word mode.

4.6 COMMANDS TO INSPECT SYSTEM OBJECTS

The inspect commands allow you to examine iRMX I objects in detail. They give specific information about the Nucleus object types. Figure 4-2 illustrates the general syntax for these commands.

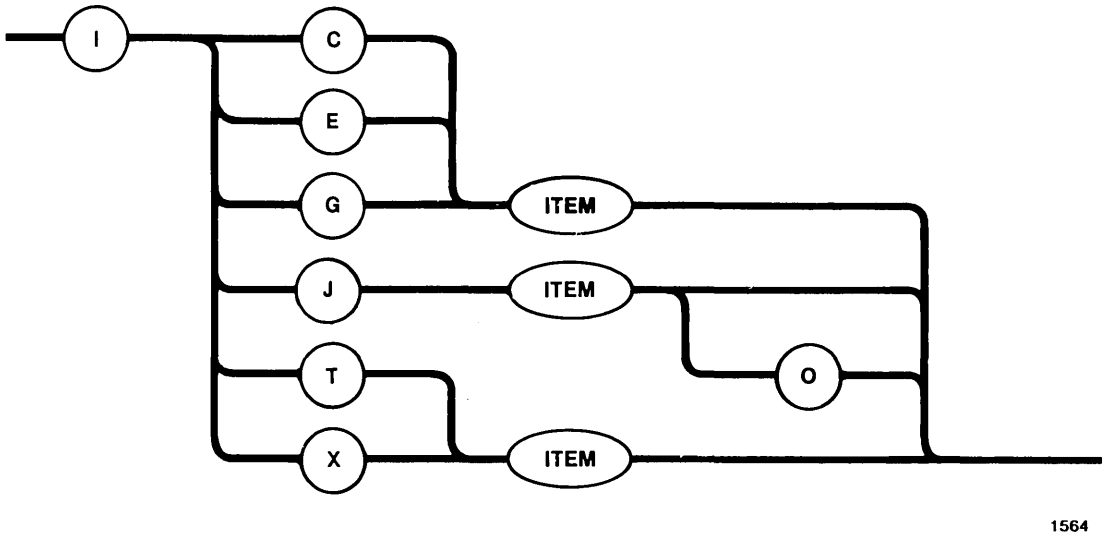


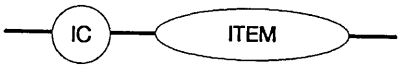
Figure 4-2. Syntax Diagram for Inspecting System Objects

The second letter of the command indicates the type of object to inspect, as follows:

C Composite
E Exchange
G Segment
J Job
T Task
X Extension

The remainder of this section describes the commands in detail.

This command displays the principal attributes of the specified composite. The syntax for the IC command is as follows:



W-1067

Parameter

ITEM — Token for the composite object to be inspected.

Description

The IC command displays the principal attributes of the composite object whose token is represented by ITEM, in the form shown in Figure 4-3.

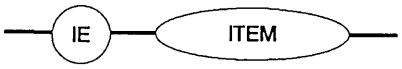
```
            ----- iRMXI COMPOSITE REPORT -----
COMPOSITE TOKEN         bbbb        CONTAINING JOB          gggg
EXTENSION TOKEN         cccc        # TOKEN SLOTS           hhhh
TOKEN(S)    ffffJ/dddde ffffJ/dddde ffffJ/dddde ffffJ/dddde

NAME(S)  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
          aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
```

Figure 4-3. An iRMX® I Composite Report®

The following describes the fields in Figure 4-3 (in alphabetical order):

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the composite is catalogued in the object directory of either the job containing the composite or the root job. If the composite is not catalogued in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the composite.
cccc	Hexadecimal token for the extension that represents license to create this type of composite.
dddd	Hexadecimal token for one of the components of the composite object.
e	Single letter that indicates the type of object dddd. This field can have any of the following values: C composite G segment J job M mailbox R region S semaphore T task X extension * a task whose stack has overflowed or whose code was loaded by the iRMX I Application Loader
ffff	Hexadecimal token for the job that contains object dddd.
gggg	Hexadecimal token for the job that contains composite object bbbb.
hhhh	Hexadecimal value specifying the maximum allowable number of component objects that the composite object can comprise.

This command displays the principal attributes of a mailbox, semaphore, or region whose token is specified. The syntax of the IE command is as follows:



W-1068

Parameter

ITEM — Token for the exchange to be inspected.

Description

The IE command displays the principal attributes of the mailbox, semaphore, or region whose token is represented by ITEM. It produces three kinds of output, one for each kind of exchange.

Mailbox Display

Figure 4-4 depicts the form of display produced by IE for a mailbox.

```
                  ----- iRMXI MAILBOX REPORT -----
MAILBOX TOKEN     bbbb              CONTAINING JOB              hhhh
# TASKS WAITING   cccc              # OBJECTS WAITING           iiii
FIRST WAITING     ddddf/eeeef       QUEUE DISCIPLINE         jjjjjjjj
CACHE SIZE        gggg

NAME(S)  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
          aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
```

Figure 4-4. An iRMX® I Mailbox Report

The following describes the fields in Figure 4-4:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the mailbox is catalogued in the object directory of either the mailbox's containing job or the root job. If the mailbox is not catalogued in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the mailbox.
cccc	Number, in hexadecimal, of tasks in the mailbox's task queue.
dddd	Token for the containing job of either the first task waiting in the task queue or the first object waiting in the object queue. Because at least one of these queues is empty, dddd is not ambiguous. If both queues are empty, dddd is absent.
eeee	Token for either the first task waiting in the task queue or the first object waiting in the object queue. Because at least one of these queues is empty, eeee is not ambiguous. If both queues are empty, eeee is 0000.
f	Single letter that indicates the type of the first task waiting in the task queue or the first object waiting in the object queue. Because at least one of these queues is empty, f is not ambiguous. If both queues are empty, f is absent. Otherwise, f has one of the following values: C composite G segment J job M mailbox R region S semaphore T task X extension
gggg	Number, in hexadecimal, of objects that the mailbox's high performance object queue can hold.
hhhh	Hexadecimal token for the job containing the mailbox.
iiii	Number, in hexadecimal, of objects in the mailbox's object queue.
jjjjjjj	Describes how waiting tasks are queued in the mailbox's task queue, either FIFO or PRIORITY.

Semaphore Display

Figure 4-5 depicts the form of the display produced by IE for a semaphore.

```
                  ----- iRMXI SEMAPHORE REPORT -----
SEMAPHORE TOKEN          bbbb       CONTAINING JOB              gggg
# TASKS WAITING          cccc       QUEUE DISCIPLINE        hhhhhhhh
CURRENT VALUE            dddd       MAXIMUM VALUE               iiii
FIRST WAITING     eeeeJ/fffft
NAME(S)      aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
             aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
```

Figure 4-5. An iRMX® I Semaphore Report

The following describes the fields in Figure 4-5:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the semaphore is catalogued in the object directory of either the semaphore's containing job or the root job. If the semaphore is not catalogued in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the semaphore.
cccc	Number, in hexadecimal, of tasks waiting in the queue.
dddd	Number, in hexadecimal, of units currently in the custody of the semaphore.
eeee	Hexadecimal token for the containing job of the first waiting task. It is absent if no tasks are waiting.
ffff	Hexadecimal token for the first waiting task. It is 0000 if no tasks are waiting.
gggg	Hexadecimal token for the semaphore's containing job.
hhhhhhhh	Describes how waiting tasks are queued in the semaphore's task queue, either FIFO or PRIORITY.
iiii	Maximum allowable number, in hexadecimal, of units that the semaphore may have in its custody.

Region Display

Figure 4-6 depicts the form of the display produced by IE for a region.

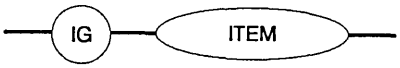
```
                  ----- iRMXI REGION REPORT -----
REGION TOKEN            bbbb        CONTAINING JOB          eeee
# TASKS WAITING         cccc        QUEUE DISCIPLINE     ffffffff
TASK IN REGION          dddd        FIRST WAITING           gggg

NAME(S)      aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
             aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
```

Figure 4-6. An iRMX® I Region Report

The following describes the fields in Figure 4-6:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the region is catalogued in the object directory of either the job containing the region or the root job. If the region is not catalogued in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the region.
cccc	Number, in hexadecimal, of tasks awaiting access to the data protected by the region.
dddd	Hexadecimal token for the task that currently has access.
eeee	Hexadecimal token for the job that contains the region.
ffffffff	Describes how waiting tasks are queued at the region, either FIFO, PRIORITY, or INVALID.

This command displays the principal attributes of the specified segment. The syntax for the IG command is as follows:



W-1069

Parameter

ITEM — Token for the segment to be inspected.

Description

The IG command displays the principal attributes of the segment whose token is represented by ITEM. Figure 4-7 depicts the form of the display produced by IG.

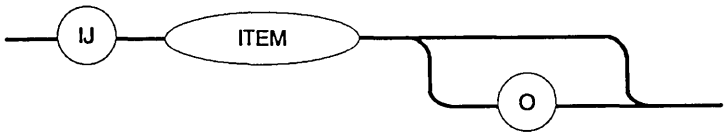
```
                    ----- iRMXI SEGMENT REPORT -----
     SEGMENT TOKEN     bbbb              CONTAINING JOB          dddd
     SEGMENT BASE      cccc              SEGMENT LENGTH         eeeee

NAME(S)  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
         aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
```

Figure 4-7. An iRMX® I Segment Report

The following describes the fields in Figure 4-7:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the segment is catalogued in the object directory of either the segment's containing job or the root job. If the segment is not catalogued in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the segment.
cccc	Base address of the segment.
dddd	Hexadecimal token for the job that contains the segment.
eeeee	Number, in hexadecimal, of bytes in the segment.

This command lists the principal attributes of a specified job. The syntax for the IJ command is as follows:



W-1075

Parameters

ITEM	A token for the job to be inspected.
O	If this option is included, the job's object directory is also listed. If omitted, the object directory is not listed.

Description

The IJ command lists the principal attributes of a job whose token is represented by ITEM. It also lists the object directory if the O option is included. If a large number of entries are in the object directory, the CONTROL-O character can be used to prevent data from rolling off the screen. The CONTROL-O special character is described in Chapter 2.

Figure 4-8 depicts the form of the display produced by the IJ command.

```
            ----- iRMXI JOB REPORT -----
JOB TOKEN                bbbb       PARENT JOB               jjjj
POOL MAXIMUM             cccc       POOL MINIMUM             kkkk
CURRENT ALLOCATED        dddd       CURRENT UNALLOCATED      llll
CURRENT # OBJECTS        eeee       CURRENT # TASKS          mmmm
MAXIMUM # OBJECTS        ffff       MAXIMUM # TASKS          nnnn
CURRENT # CHILD JOBS     gggg       DELETION PENDING          ppp
EXCEPTION MODE           hhhh       EXCEPTION HANDLER  qqqq:rrrr
MAXIMUM PRIORITY         iiii

NAME(S)  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
         aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa

            ----- OBJECT DIRECTORY -----
MAXIMUM SIZE             uuuu        VALID ENTRIES          vvvv
NAME         TOKEN      NAME         TOKEN      NAME         TOKEN
sssssssssss tttt       sssssssssss tttt       sssssssssss tttt
```

Figure 4-8. An iRMX® I Job Report

The following describes the fields in Figure 4-8:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the job is catalogued in the object directory of either the job's parent job or the root job. If the job is not catalogued in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the job.
cccc	Maximum number, in hexadecimal, of 16-byte paragraphs that the job's pool can contain.
dddd	Number of paragraphs either allocated to tasks in the job or lent to child jobs.
eeee	Number, in hexadecimal, of existing objects in job bbbb.
ffff	Maximum number, in hexadecimal, of objects that can exist simultaneously in job bbbb.
gggg	Number, in hexadecimal, of existing jobs that are offspring of job bbbb.

hhhh — Exception mode for the job's default exception handler. Possible values are as follows:

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

iiii — Hexadecimal value that indicates the maximum (numerically lowest) allowable priority for tasks in the job.

jjjj — Hexadecimal token for the parent of job bbbb. If job bbbb is the root job, however, jjjj is "ROOT".

kkkk — Minimum number, in hexadecimal, of 16-byte paragraphs that the job's pool can contain.

llll — Number, in hexadecimal, of unused 16-byte paragraphs in the job's initial pool.

mmmm — Number, in hexadecimal, of tasks currently in the job.

nnnn — Maximum number, in hexadecimal, of tasks that can exist simultaneously in job bbbb.

ppp — Indicator which tells whether a task has attempted to delete the job but was unsuccessful because the job has obtained protection from the DISABLE$DELETION system call. The possible values of ppp are YES and NO.

qqqq — Base, in hexadecimal, of the start address of the job's default exception handler.

rrrr — Hexadecimal offset, relative to qqqq, of the start address of the job's default exception handler.

ssssssssssss — Each such field contains the name under which an object is catalogued in the job's object directory. If no entries are in the object directory, these fields are blank.

tttt — Each such field contains a token, in hexadecimal, of the object whose name (in the directory) appears next to it.

uuuu	Maximum allowable number, in hexadecimal, of entries in the job's object directory.
vvvv	Number, in hexadecimal, of entries currently in the job's object directory.

This command lists the principal attributes of a specified task. The syntax for the IT command is as follows:

W-1070

Parameter

ITEM — Token for the task to be inspected.

Description

The IT command displays the principal attributes of the task whose token is represented by ITEM. Figure 4-9 depicts the form of display produced by IT.

```
            ----- iRMXI TASK REPORT -----
TASK TOKEN                bbbb     CONTAINING JOB              kkkk
STACK SEGMENT BASE        cccc     STACK SEGMENT OFFSET        llll
STACK SEGMENT SIZE        dddd     STACK SEGMENT LEFT          mmmm
CODE SEGMENT BASE         eeee     DATA SEGMENT BASE           nnnn
INSTRUCTION POINTER       ffff     TASK STATE               pppppppp
STATIC PRIORITY           gggg     DYNAMIC PRIORITY            qqqq
SUSPENSION DEPTH          hhhh     SLEEP UNITS REQUESTED       rrrr
EXCEPTION MODE            iiii     EXCEPTION HANDLER       ssss:tttt
NPX TASK                  jjj

NAME(S)  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
         aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
```

Figure 4-9. An iRMX® I Task Report

The following describes the fields in Figure 4-9:

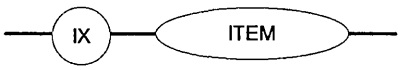
Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the task is catalogued in the object directory of either the task's containing job or the root job. If the job is not catalogued in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the task.
cccc	Base address, in hexadecimal, of the task's stack segment.
dddd	Size, in bytes, of the task's stack segment.
eeee	Base address, in hexadecimal, of the task's code segment.
ffff	Current value, in hexadecimal, of the task's instruction pointer.
gggg	Hexadecimal priority of the task.
hhhh	Current number, in hexadecimal, of "suspends" against the task. Before the task can be made ready, each "suspend" must be countered with a "resume".
iiii	Exception mode for the task's exception handler. Possible values are as follows:

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

jjj	Indicator that tells whether the task uses the NPX. The possible values of are YES and NO.
kkkk	Hexadecimal token for the task's containing job.
llll	Hexadecimal offset, relative to cccc, of the task's stack segment.
mmmm	Hexadecimal number of bytes currently available in the task's stack.
nnnn	Base address, in hexadecimal, of the task's data segment.
pppppppp	Current execution state of the task. Possible values are "READY," "ASLEEP," "SUSPENDED," "ASLEEP/SUSP," "BROKEN," and "INVALID."
qqqq	A temporary hexadecimal priority sometimes assigned to the task by the Nucleus to improve system performance.

rrrr	If the task is asleep or asleep/suspended, this is the number of sleep units that the task requested before going to sleep. If the task is ready or suspended, qqqq is 0000.
ssss	Base, in hexadecimal, of the start address of the task's exception handler.
tttt	Hexadecimal offset, relative to ssss, of the start address of the task's exception handler.

This command displays the principal attributes of the specified extension object. The syntax for the IX command is as follows:



W-1071

Parameter

ITEM — Token for the extension object to be inspected.

Description

The IX command displays the principal attributes of the extension whose token is represented by ITEM. Figure 4-10 depicts the form of the display produced by IX.

```
            ----- iRMXI EXTENSION REPORT -----
EXTENSION TOKEN         bbbb          CONTAINING JOB       dddd
TYPE CODE               cccc          DELETION MAILBOX     eeee

NAME(S)  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
         aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
```

Figure 4-10. An iRMX® I Extension Report

The following describes the fields in Figure 4-10:

Field	Meaning
aaaaaaaaaaa	Each such field contains a name under which the extension is catalogued in the object directory of either the job containing the extension or the root job. If the extension is not catalogued in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the extension.
cccc	Hexadecimal type code associated with composite objects licensed by this extension.
dddd	Hexadecimal token for the job containing this extension.
eeee	Hexadecimal token for the deletion mailbox associated with the extension. If there is no deletion mailbox for the extension, "NONE" is displayed here.

4.7 COMMANDS TO VIEW OBJECT LISTS

These commands enable you to view lists of iRMX I objects. Figure 4-11 illustrates the general syntax for commands in this section.

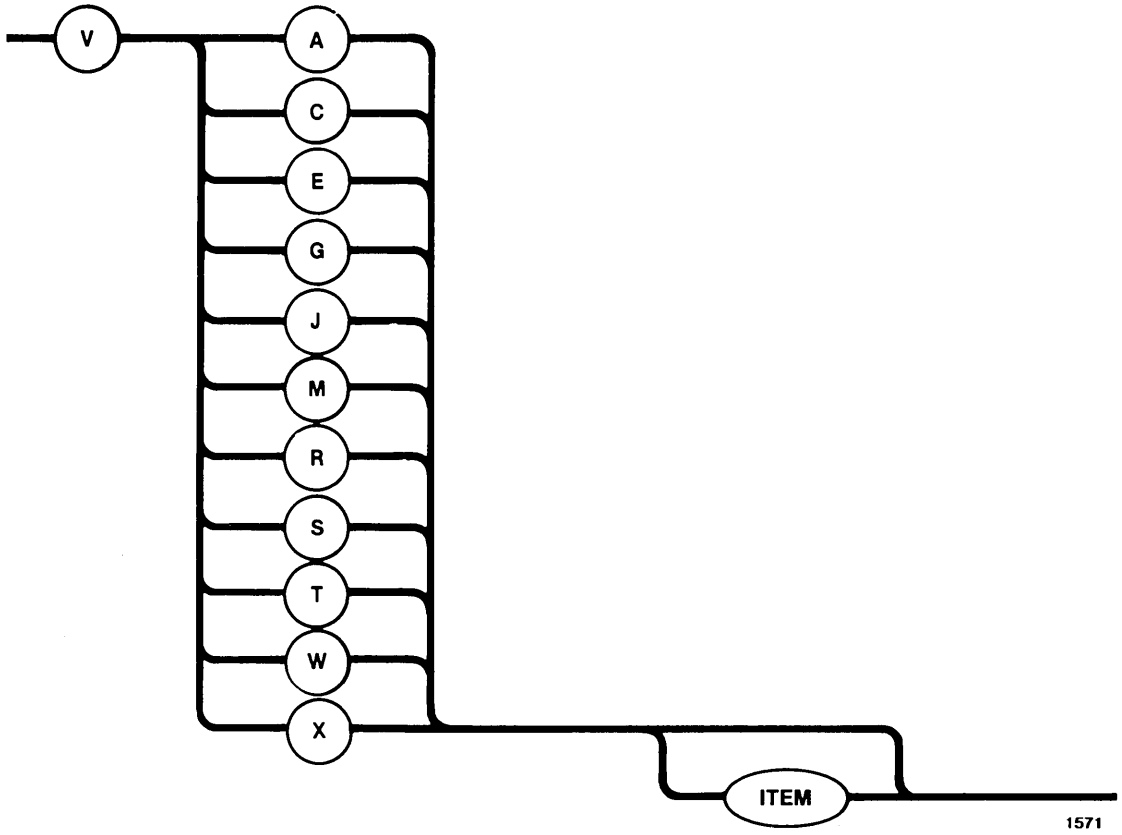


Figure 4-11. Syntax Diagram for Viewing iRMX® I Object Lists

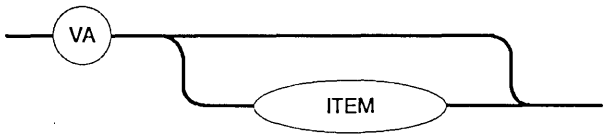
The second letter of the command indicates the type of object list to display, as follows:

A	Asleep tasks
C	Composites
E	Exchanges
G	Segments
J	Jobs
M	Mailbox queues
R	Ready tasks
S	Suspended tasks
T	Tasks
W	Waiting task queues
X	Extensions

The remainder of this section describes the commands in detail.

VIEWING THE ASLEEP TASKS -- VA

This command displays a list of asleep tasks. The syntax for the VA command is as follows:



W-1092

Parameter

ITEM	Token for a job. If this option is included, the Debugger lists only those asleep tasks that are contained in the specified job. If this option is omitted, all asleep tasks in the system are listed.

Description

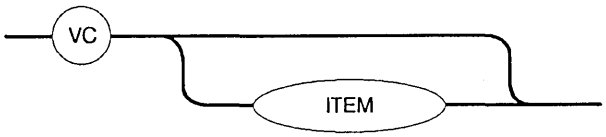
The VA command displays asleep tasks as

```
SA = jjjjJ/ttttT  jjjjJ/ttttT ... jjjjJ/ttttT
```

where:

tttt	Token of an asleep task.
jjjj	Token for the job containing the task. An asterisk following the task token indicates that the task has overflowed its stack.

This command displays a list of composite objects. The syntax for the VC command is as follows:



W-1093

Parameter

ITEM — Token for a job. If this option is included, the Debugger lists only the composite objects contained in the specified job. If this option is omitted, all composite objects in the system are displayed.

Description

The VC command displays composite objects as

```
CL = jjjjJ/ccccC jjjjJ/ccccC ... jjjjJ/ccccC
```

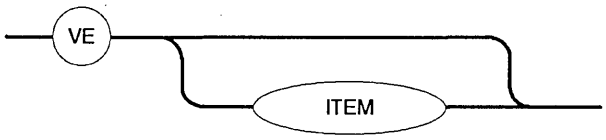
where:

cccc — Token for a composite object.

jjjj — Token for the job containing the composite object.

VIEWING EXCHANGES -- VE

This command displays a list of exchanges. The syntax for the VE command is as follows:



W-1094

Parameter

ITEM	Token for a job. If this option is included, the Debugger lists only those exchanges contained in the specified job. If this option is omitted, all exchanges in the system are listed.

Description

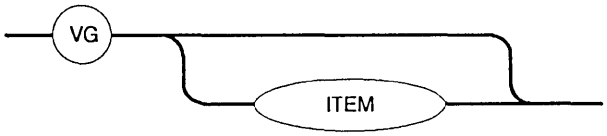
The VE command lists exchanges as

```
EL = jjjjJ/xxxxt jjjjJ/xxxxt ... jjjjJ/xxxxt
```

where:

xxxx	Token for an exchange.
t	Type of the exchange (M for mailbox, S for semaphore, or R for region).
jjjj	Token for the job containing the exchange.

This command displays a list of segments. The syntax for the VG command is as follows:



W-1095

Parameter

ITEM	Token for a job. If this option is included, the Debugger lists only the segments contained in the specified job. If this option is omitted, all segments in the system are displayed.

Description

The VG command displays segments as

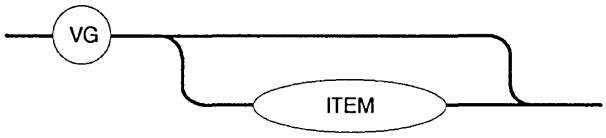
```
GL = jjjjJ/ggggG jjjjJ/ggggG ... jjjjJ/ggggG
```

where:

gggg	Token for a segment.
jjjj	Token for the job containing the segment.

VIEWING JOBS -- VJ

This command displays a list of jobs. The syntax for the VJ command is as follows:



W-1095

Parameter

ITEM	Token for a job. If this option is included, the Debugger lists only those jobs that are children of the specified job. If this option is omitted, all jobs in the system are listed.

Description

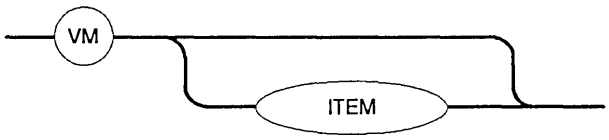
The VJ command displays jobs as

```
JL = ppppJ/jjjjJ ppppJ/jjjjJ ... ppppJ/jjjjJ
```

where:

jjjj	Job token.
pppp	Token of its parent job. If the job designated by jjjj is the root job, then "ROOT" replaces "ppppJ".

This command displays object queues at mailboxes. The syntax for the VM command is as follows:



W-1097

Parameter

ITEM — Token for a mailbox or a job. If you specify a mailbox token for this option, the Debugger lists only the object queue associated with the specified mailbox. If you specify a job token for this option, the Debugger lists all object queues in the specified job. If you omit this option, the Debugger displays object queues for all exchanges in the system.

Description

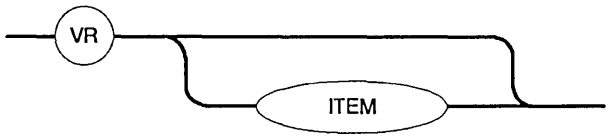
The VM command displays object queues at mailboxes as

```
ML jjjjJ/mmmmM = jjjjJ/oooot jjjjJ/oooot ... jjjjJ/oooot
ML jjjjJ/mmmmM = jjjjJ/oooot jjjjJ/oooot ... jjjjJ/oooot
                      .
                      .
                      .
ML jjjjJ/mmmmM = jjjjJ/oooot jjjjJ/oooot ... jjjjJ/oooot
```

where:

mmmm	Token for a mailbox.
oooo	Token for an object in that mailbox's object queue.
t	Type of the object (J for job, T for task, M for mailbox, S for semaphore, and G for segment).
jjjj	Token for the job containing the mailbox or object.

This command displays a list of ready tasks. The syntax for the VR command is as follows:



W-1098

Parameter

ITEM — Token for a job. If this option is included, the Debugger lists, in priority order, the ready tasks contained in the specified job. If this option is omitted, all ready tasks in the system are listed in order of priority.

Description

The VR command displays ready tasks as

```
RL = jjjjJ/ttttT jjjjJ/ttttT ... jjjjJ/ttttT
```

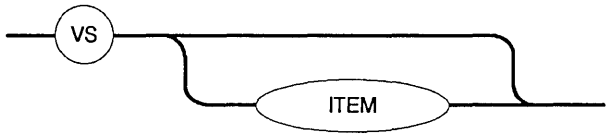
where:

tttt — Token of a ready task.

jjjj — Token for the job containing the task. An asterisk following a task token indicates that the task has overflowed its stack.

VIEWING SUSPENDED TASKS -- VS

This command displays a list of suspended tasks. The syntax for the VS command is as follows:



W-1099

Parameter

ITEM	Token for a job. If this option is included, the Debugger lists only those suspended tasks that are contained in the specified job. If this option is omitted, all suspended tasks in the system are listed.

Description

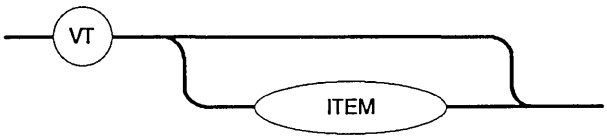
The VS command displays suspended tasks as

```
SL = jjjjJ/ttttT jjjjJ/ttttT ... jjjjJ/ttttT
```

where:

tttt	Token of a suspended task.
jjjj	Token for the job containing the task. An asterisk following a task token indicates that the task has overflowed its stack.

This command displays a list of tasks. The syntax for the VT command is as follows:



W-1100

Parameter

ITEM	Token for a job. If this option is included, the Debugger lists only those tasks contained in the specified job. If this option is omitted, all tasks in the system are listed.

Description

The VT command displays tasks as

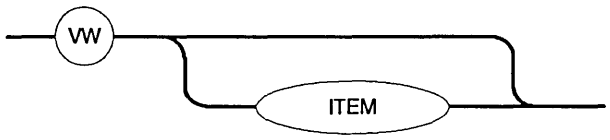
```
TL = jjjjJ/ttttT jjjjJ/ttttT ... jjjjJ/ttttT
```

where:

tttt	Task token.
jjjj	Token for the job that contains the task. An asterisk following a task token indicates that the task has overflowed its stack.

VIEWING WAITING TASK QUEUES -- VW

This command displays the waiting task queues at exchanges. The syntax for the VW command is as follows:



W-1101

Parameter

ITEM	Token for an exchange or a job. If you specify an exchange token for this option, the Debugger lists only the task queue associated with the specified exchange. If you specify a job token for this option, the Debugger lists all task queues in the specified job. If you omit this option, the Debugger displays task queues for all exchanges in the system.

Description

The VW command displays task queues at exchanges as

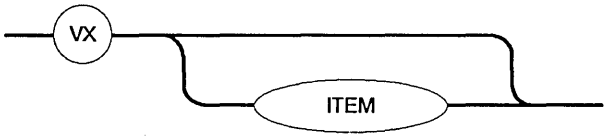
```
WL jjjjJ/xxxxt = jjjjJ/ttttT jjjjJ/ttttT ... jjjjJ/ttttT
WL jjjjJ/xxxxt = jjjjJ/ttttT jjjjJ/ttttT ... jjjjJ/ttttT
                      .
                      .
                      .
WL jjjjJ/xxxxt = jjjjJ/ttttT jjjjJ/ttttT ... jjjjJ/ttttT
```

where:

xxxx	Token for an exchange.
t	Type of the exchange (M for mailbox, S for semaphore 4, or R for region).
tttt	Token for a task queued at that exchange.
jjjj	Token for the job containing the task. An asterisk indicates that either the task has overflowed its stack or the task was loaded by the Application Loader.

VIEWING EXTENSIONS -- VX

This command displays either a list of extension objects or a list of composite objects associated with a particular extension object. The syntax for the VX command is as follows:



W-1102

Parameter

ITEM	Token for an extension object. If this option is included, the Debugger lists all composite objects associated with the specified extension object. If this object is omitted, the Debugger lists all extension objects in the system.

Description

If the ITEM parameter is omitted, the VX command displays extension objects as follows:

```
XL = jjjjJ/xxxxX jjjjJ/xxxxX ... jjjjJ/xxxxX
```

where:

xxxx	Token for an extension object.
jjjj	Token for the job containing the extension.

If the ITEM option is included, the VX command lists the composite objects associated with a particular extension object as follows:

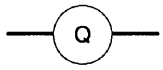
```
XL jjjjJ/xxxxX = kkkkJ/ccccC kkkkJ/ccccC ... kkkkJ/ccccC
```

where:

xxxx	Token for the extension object.
jjjj	Token for the job containing the extension.
cccc	Token for the composite object associated with the specified extension.
kkkk	Token for the job containing the composite object.

EXITING THE DEBUGGER -- Q

This command exits the Debugger. The syntax for the Q command is as follows:



W-1072

Description

The Q command deactivates the Debugger. When a debugging session is terminated, the tables and lists the Debugger maintains are not destroyed. Q also displays the message "EXIT iRMX I DEBUGGER."

CONFIGURATION 5

5.1 INTRODUCTION TO DEBUGGER CONFIGURATION

The Debugger is a configurable layer of the operating system. It contains several options that you can adjust to meet your specific needs. To make configuration choices, Intel provides three kinds of information:

- A list of configurable options
- Detailed information about the options
- Procedures to enable you to specify your choices

The balance of this chapter provides the first category of information. To obtain the second and third categories of information, refer to the *Guide to the iRMX® I Interactive Configuration Utility.*

Debugger configuration is almost identical to Terminal Handler configuration (except that only one Debugger can be present in the application system). Debugger configuration involves selecting characteristics of the Debugger's Terminal Handler and specifying information about the processor board and the terminal. The following sections describe the configurable options available on the Debugger.

5.2 BAUD RATE

You can set the baud rate for the Debugger's Terminal Handler to any of the following values:

110	600	4800
150	1200	9600
300	2400	19200

The default baud rate for the Debugger's Terminal Handler is 9600.

5.3 BAUD COUNT

The baud count calculates internal timer values, given the clock input frequency. The baud count sets the limits on the baud rate attributes of the Debugger's Terminal Handler. If your system's programmable interval timer (PIT) has a clock input frequency other than 1.2288 MHz, you must set the baud count. The default value for the baud count is 4.

5.4 RUBOUT MODE AND BLANKING CHARACTER

You can delete a character from the buffer in one of two ways:

- Echo mode
- Replace mode

In the echo mode, the character being deleted from the current line is re-echoed to the display. For example, entering "CAT" and then pressing RUBOUT three times results in the display "CATTAC".

In the replace mode, the deleted character is replaced on the display with the blanking character. For example, entering "CAT" and then pressing RUBOUT three times deletes "CAT" from the display.

The blanking-character and the default RUBOUT mode can be specified when you generate your system. If they are not specified, they default to a blank (20H)and echo.

5.5 USART

The USART is a device that, depending on the application, can either convert serial data to parallel data or convert parallel data to serial data. The Debugger's Terminal Handler requires an 8251A USART as a terminal controller. You must specify

- The port address of the USART (default value is 0D8H).
- The interval between the port addresses for the USART.
- The number of bits of valid data per character that can be sent from the USART, (default value is 7).

5.6 PIT

You must specify the following information about the programmable interval timer (PIT):

- The port address of the PIT, (default value is 0D0H).
- The interval between the port addresses for the PIT.
- The number of the PIT counter connected to the USART clock input, (default value is 2).

5.7 MAILBOX NAMES

You can change the default names of both the input mailbox (RQTHNORMIN) and the output mailbox (RQTHNORMOUT). The new names must not be over 12 alphanumeric characters long.

5.8 INTERRUPT LEVELS

You must specify the interrupt levels used by the Debugger's Terminal Handler for input and output. You choose interrupt levels by selecting a value that corresponds to a particular interrupt value. The default value for the input interrupt level is 68H, and the default value for the output interrupt level is 78H.

ERROR MESSAGES A

A.1 INTRODUCTION TO DEBUGGER ERROR MESSAGES

This appendix lists the error messages that can occur when you enter Debugger commands. Since the Debugger reads commands on a line-by-line basis, it will not issue an error message for a command until you terminate the command with an end-of-line character (CARRIAGE RETURN or LINE FEED). Then, if the Debugger detects an error, it generates a display of the following form:CARRIAGE RETURNLINE FEED

```
command portion #
error message
```

where "command portion" consists of the command up to the point where the Debugger detected the error, and "error message" consists of one of the following:

Message	Description
ATTEMPT TO MODIFY NON-RAM LOCATION	You tried to define a breakpoint at a non-RAM memory location.
BREAKPOINT TASK NOT AN NPX TASK	You specified the N command, but the breakpoint task was not designated as a Numeric Processor Extension (NPX) task at its creation.
COMMAND TOO COMPLEX	To process your commands, the Debugger maintains a semantic stack, where it places all the semantic entities of the commands. Your command was too complex and overflowed this stack. To correct this problem, you should first define numeric variables for some of the more complex expressions, and then use these variables in your command in place of the expressions.
DEBUGGER POOL TOO SMALL	To process your command, the Debugger tried to create an iRMX I segment, but not enough free space was available in the system to create this segment.
DUPLICATE SYMBOL	You attempted to define a numeric or breakpoint variable name that was already defined.

Message	Description
EXECUTION BREAKPOINT ALREADY DEFINED	You attempted to define (or redefine) an execution breakpoint at an address that already specifies an execution breakpoint. This breakpoint may have been set up by the Debugger or by the Monitor and must be deleted before a new one can use this location.
INTERRUPT TASK NOT ON BREAKPOINT LIST	You attempted to make an interrupt task the current breakpoint task without first suspending that interrupt task. An interrupt task can be made the current breakpoint task only by first incurring a breakpoint.
INVALID TASK STATE	The Nucleus-maintained task descriptor contains inconsistent information. You may have overwritten this area of memory. The task probably will not continue to run.
INVALID TOKEN	You specified a token for a different kind of object than the command required.
ITEM NOT FOUND	You tried to delete or change a nonexistent numeric variable.
NO BREAKPOINT TASK	You entered the R or N command without first establishing a breakpoint task.
SYNTAX ERROR	The command is syntactically incorrect.
TASK NOT ON BREAKPOINT LIST	You tried to remove a task from the breakpoint list with the G command when the task was not on the list.
TASK NOT SUSPENDABLE. WILL BE BROKEN WHEN SUSPENDABLE	You entered the BT command to establish a breakpoint task, but the Debugger could not suspend the task in its current state (for example, the task currently has access to a region). The Debugger will suspend the task when it becomes possible to do this.
UNDEFINED SYMBOL	The Debugger was unable to find the specified symbol in the local symbol table, the object directory of the breakpoint task's job, or the root object directory.
UNKNOWN BREAKPOINT iAPX 86, 88 MONITOR NOT CONFIGURED	The Debugger encountered a breakpoint for which it had no record. It tried to pass the breakpoint to the Monitor but could not because the Monitor is not included in your system.

INDEX

A

B

C

D

E

G

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative.

1. Please describe any errors you found in this publication (include page number).

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

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

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

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____ PHONE ( ) _____

CITY _____ STATE _____ ZIP CODE _____

(COUNTRY)

Please check here if you require a written reply ☐

VE'D LIKE YOUR COMMENTS . . .

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

you are in the United States, use the preprinted address provided on this form to return your ɔmments. No postage is required. If you are not in the United States, return your comments to the Intel ales office in your country. For your convenience, international sales office addresses are printed on ıe last page of this document.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 HILLSBORO, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
OMSO Technical Publications, MS: HF3-72
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124-9978

INTERNATIONAL SALES OFFICES

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

BELGIUM
Intel Corporation SA
Rue des Cottages 65
B-1180 Brussels

DENMARK
Intel Denmark A/S
Glentevej 61-3rd Floor
dk-2400 Copenhagen

ENGLAND
Intel Corporation (U.K.) LTD.
Piper's Way
Swindon, Wiltshire SN3 1RJ

FINLAND
Intel Finland OY
Ruosilante 2
00390 Helsinki

FRANCE
Intel Paris
1 Rue Edison-BP 303
78054 St.-Quentin-en-Yvelines Cedex

ISRAEL
Intel Semiconductors LTD.
Atidim Industrial Park
Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

ITALY
Intel Corporation S.P.A.
Milandfiori, Palazzo E/4
20090 Assago (Milano)

JAPAN
Intel Japan K.K.
Flower-Hill Shin-machi
1-23-9, Shinmachi
Setagaya-ku, Tokyo 15

NETHERLANDS
Intel Semiconductor (Netherland B.V.)
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam

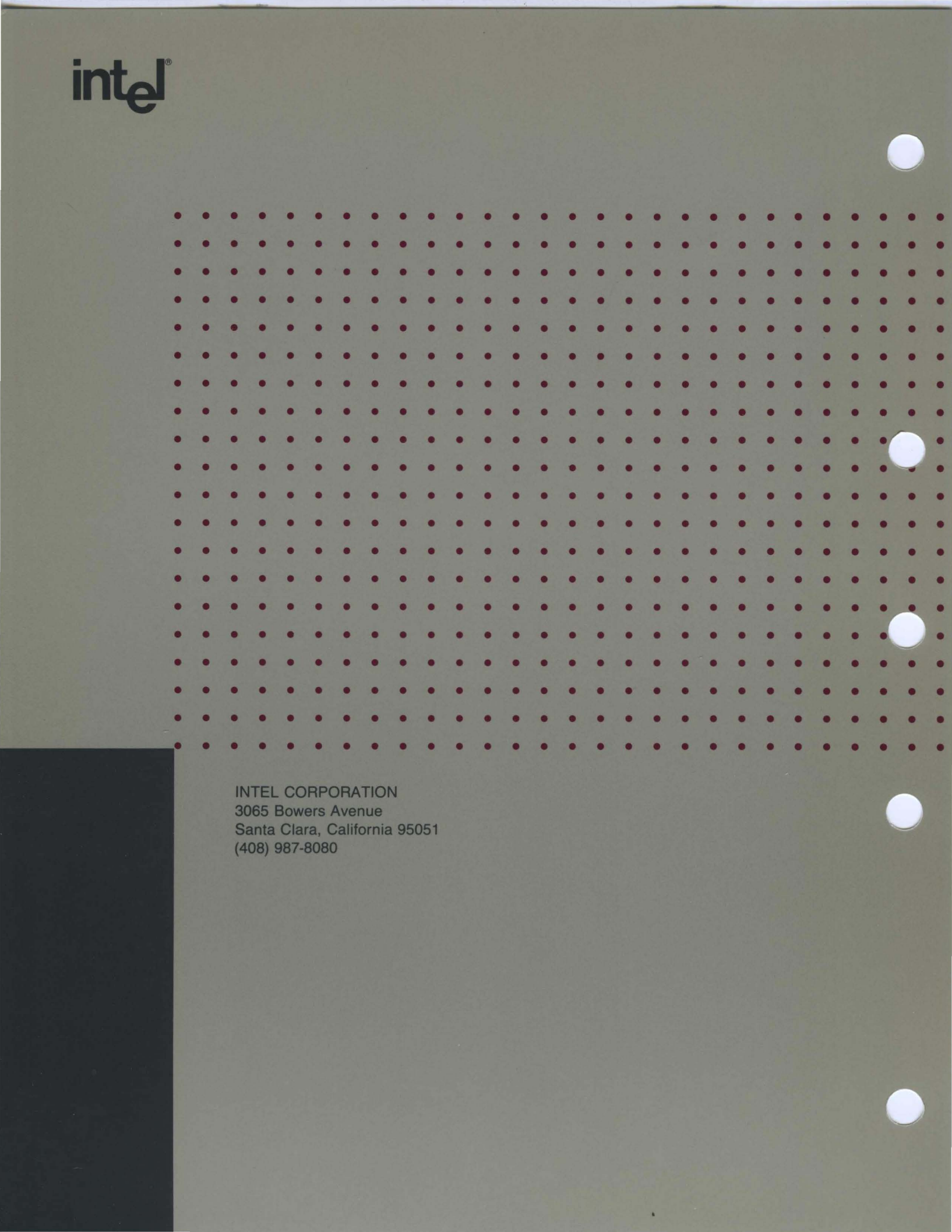
NORWAY
Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten

SPAIN
Intel Iberia
Calle Zurbaran 28-IZQDA
28010 Madrid

SWEDEN
Intel Sweden A.B.
Dalvaegen 24
S-171 36 Solna

SWITZERLAND
Intel Semiconductor A.G.
Talackerstrasse 17
8125 Glattbrugg
CH-8065 Zurich

WEST GERMANY
Intel Semiconductor G.N.B.H.
Seidlestrasse 27
D-8000 Munchen



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