iRMX 86™ DEBUGGER REFERENCE MANUAL

Manual Order No. 143323-001

REV.	REVISION HISTORY	PRINT DATE
-001	Original Issue. Contains information formerly found in iRMX 86 NUCLEUS, TERMINAL HANDLER, AND DEBUGGER REFERENCE MANUAL and iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL. This issue revises format, corrects technical and typographical errors, and describes the N and VA commands.	5/81

A365/681/4K NCG

PREFACE

This manual documents the Debugger, a subsystem of the iRMX 86 Operating System that allows you to interactively examine your application system in order to find and correct errors. It contains introductory and overview material, as well as detailed descriptions of all 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 86 NUCLEUS REFERENCE MANUAL. System programmers should also be familiar with the concepts introduced in the iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL.

CONVENTIONS

This manual contains several examples of Debugger commands entered at a terminal. In these examples, user input is underscored to distinguish it from Debugger output.

RELATED PUBLICATIONS

The following manuals provide additional background and reference information.

Manual	Number
Introduction to the iRMX 86 Operating System	9803124
iRMX 86 Nucleus Reference Manual	9803122
iRMX 86 Terminal Handler Reference Manual	143324
iRMX 86 System Programmer's Reference Manual	142721

CONTENTS

CONTENTS (continued)

CHAPTER 1. INTRODUCTION

The development of almost every software application requires debugging. To aid in the development of iRMX 86 application systems, Intel provides three debugging tools. One, the iRMX 86 Debugger, is a powerful tool which is sensitive to the data structures that the Nucleus maintains. The other debugging tools are the ICE-86 In-Circuit Emulator and the iSBC 957A monitor.

The Debugger supplies its own Terminal Handler, which includes all of the capabilities described in the iRMX 86 TERMINAL HANDLER REFERENCE MANUAL. Your application software can make use of the Debugger's Terminal Handler, or you can include a separate version (or versions) of the Terminal Handler in your system configuration for application use. Refer to the iRMX 86 CONFIGURATION GUIDE for further configuration information.

DEBUGGER CAPABILITIES

The iRMX 86 Debugger enables you to do the following:

- Use the Debugger as a task, job, or system exception handler.

- View iRMX 86 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 86 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.

- Deactivate the Debugger.

DEBUGGING CAPABILITIES IN THE ICE-86 EMULATOR

The ICE-86 In-Circuit Emulator provides several debugging capabilities.
In particular, an ICE-86 emulator lets you:

- Get closer to the hardware level by examining the contents of
  input pins and input ports. You can also change the values at
  output ports.

- Set breakpoints.

- Use memory in your Intel Microcomputer Development System as if
  it were on your prototype board.

- Look at the most recent 80 to 150 assembly language instructions
  executed.

To learn more about the ICE-86 Emulator, consult the ICE-86 IN-CIRCUIT
EMULATOR OPERATING INSTRUCTIONS FOR ISIS-II USERS.

DEBUGGING CAPABILITIES IN THE iSBC 957A MONITOR

The iSBC 957A monitor is used with the iSBC 86/12A single board computer
to provide several capabilities that can aid you in debugging. With the
monitor, you can do the following:

- Examine and modify the contents of iAPX 86 registers and
  absolute memory locations.

- Set breakpoints.

- Single step program execution.

- Do I/O to and from ports.

- Move or compare blocks of memory.

To learn more about the iSBC 957A monitor, refer to the iSBC 957A
INTELLEC -- iSBC 86/12A INTERFACE AND EXECUTION PACKAGE USER'S GUIDE.

REASONS FOR USING THE iRMX 86 DEBUGGER

There are three important reasons for using the iRMX 86 Debugger instead
of (or in addition to) the other available debuggers. They are:

- The iRMX 86 Debugger is aware of the data structures associated
  with the Nucleus and the Nucleus objects. The other debuggers
  can examine memory only as data. The iRMX 86 Debugger
  interprets some of that data so that it can provide lists of
  iRMX 86 objects, as well as detailed information about

individual objects. Because of this feature, you do not have to sort through complicated data structures in order to obtain information about objects.

- The iRMX 86 Debugger allows you to manipulate (or examine) one task while other tasks in the system continue to run. With the other debuggers, encountering a breakpoint stops the processor. With the iRMX 86 Debugger, encountering a breakpoint stops only a single task (or possibly the tasks in a single job). Other tasks can continue to run. This feature helps you debug in a multitask environment when the interactions between individual tasks are important.

- The iRMX 86 Debugger allows you to monitor system activity without interfering in the execution. You can, for example, set a breakpoint at a mailbox so that the Debugger displays information each time a task sends to or receives from the mailbox. You have the option of doing this without stopping the task or interfering with it in any way. This feature helps you verify that certain system activities do indeed occur.

INVOKING THE DEBUGGER

You can invoke the Debugger from your iRMX 86 terminal by entering:

    control-D

The Debugger responds with its sign-on message:

    iRMX 86 DEBUGGER V3.0
    *

The asterisk is the prompt character for the Debugger. It indicates that the Debugger is ready to accept additional input from the terminal.

In addition to the functions the Debugger can perform when it has been invoked, there are two services it can perform at any time, even when not invoked. First, if a task encounters a breakpoint, the Debugger responds as described in Chapter 4.

Second, 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. You can use the %SYSTEM macro during configuration to specify the Debugger as the system default exception handler. Unless this default is overridden at job or task creation time, the Debugger automatically becomes the task's exception handler. Refer to the iRMX 86 CONFIGURATION GUIDE for a description of the %SYSTEM macro.

● By having the Debugger declared as the exception handler when
  the task is created with CREATE$JOB or CREATE$TASK. An example
  of code setting up one of these calls is the following:

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

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

CHAPTER 2. 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.

END-OF-LINE CHARACTERS

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

    Carriage Return
    Line Feed
    ESCape

Both Carriage Return and Line Feed send the current input line to the
Debugger for processing. ESCape causes the Debugger to discard the
current input line and display a prompt.

CONTROL-S

The Debugger generates display at the iRMX 86 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.

CONTROL-Q

Control-Q negates the effect of a previously entered control-S
character. To resume the ouput 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.

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.

2-1

CONTROL-D

Occasionally you will want to terminate a Debugger memory command
function response before it is finished. For example, if you asked for a
display of memory locations 0000H to 0FFFFH, it would be natural to
change your mind. To abort the display and regain the Debugger prompt,
enter control-D.

Note that control-O affects the display only, whereas control-D stops the
function entirely.

When using the iRMX 86 Debugger, you sit at a terminal and type
commands. This chapter describes the syntactical standards for commands
to the Debugger, and it introduces notational conventions that are used
throughout this manual.

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. The
  key letters or pairs of letters are BL, BT, D, DB, G, I, L, M,
  N, Q, R, V, and Z.

- In a few cases, a command is specified by beginning the command
  with a name. A name, for the Debugger, must consist 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 as delimiters
between elements of a command; they are mandatory except as follows:

- Immediately after a command key that is not a name.

- Between a letter or digit and a non-letter, non-digit. The
  legal characters of the latter type are the following: ; @ =
  / \ : ( ) * + - ,

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 setup, 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:
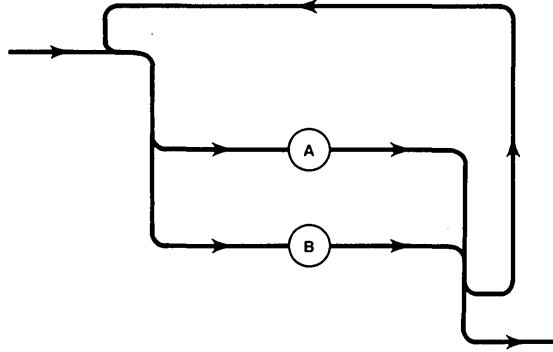
If such a string has to begin with an A, the schematic could be drawn as:
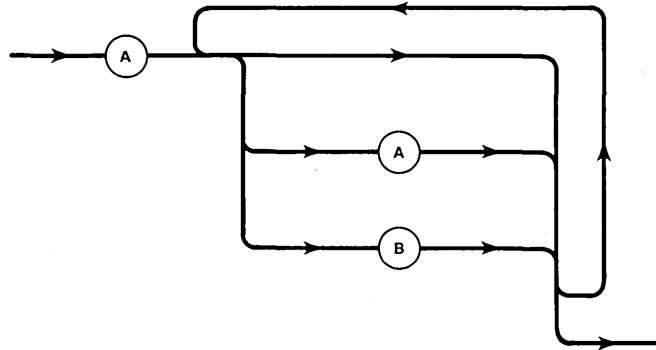
In the second drawing, it is necessary to represent the letter A twice because A is playing two roles. It is the first symbol (necessarily) and it is a symbol that may (optionally) be used after the first symbol. Note that a train could avoid the second A but cannot avoid the first A. The arrows are not necessary and henceforth are omitted.

SPECIAL SYMBOLS FOR THE DEBUGGER

The entities that will be used in the remainder of this manual, as A and B were used in the previous paragraph, are the following:

- CONSTANT. Constants are always hexadecimal. Unlike such constants in PL/M-86, they do not require an H as the last character. H's may be used if desired. Leading zeroes are not necessary unless they help to distinguish between constants and other things. For example, AH is a register in the iAPX 86, 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

● EXPRESSION. As in algebra, an expression is either a term or is
  the result of adding and subtracting terms. Also as in algebra,
  a term is a product; each factor in the product is either 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
  3-1:

                              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.

● ITEM. An item is either an expression or one of the segment
  registers of the iAPX 86 microprocessor. The values of items
  are used variously as tokens and as offsets in Debugger
  commands. Graphically, an item is defined in Figure 3-2.

TERM:

* CONSTANT NAME (EXPRESSION) AX BX CX DX SI DS ES CS SS IP FL DI BP SP

EXPRESSION:

TERM + − TERM
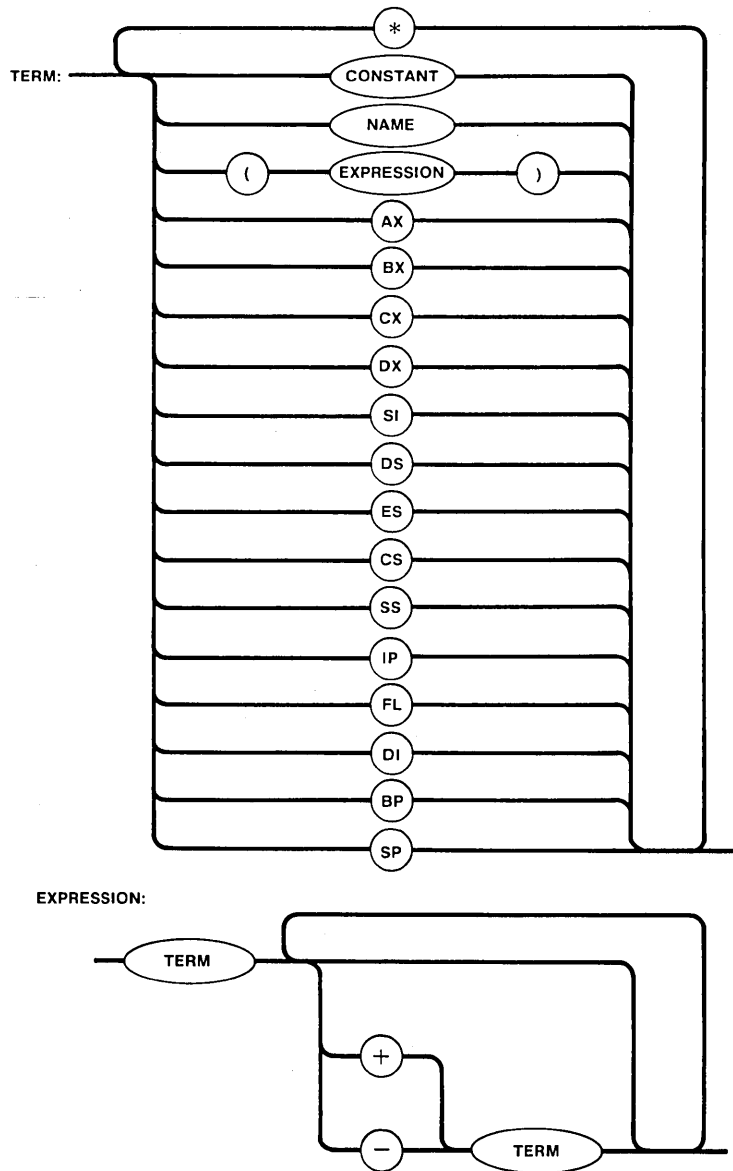
Figure 3-1. Syntax Diagrams for Term and Expression

ITEM:

EXPRESSION CS DS ES SS
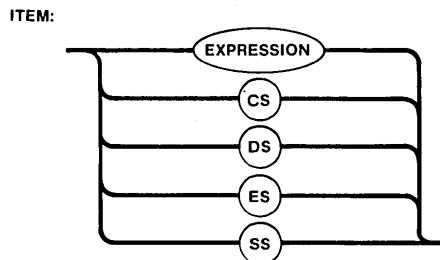
Figure 3-2. Syntax Diagram for Item

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 groupings are as follows:

    Symbolic Name Commands
    Breakpoint Commands
    Memory Commands
    Commands to Inspect iRMX 86 Objects
    Commands to View Object Lists
    Commands to Exit the Debugger

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

Between this introduction and the discussions of the individual commands
is a command dictionary. This dictionary, which lists the commands in
alphabetical order, includes short descriptions and page numbers of the
complete descriptions in this chapter. Those commands which are not
associated with specific command letters are listed at the end of the
dictionary.

COMMAND DICTIONARY

SYMBOLIC NAME COMMANDS

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

- A Debugger-maintained symbol table. The table contains name/value pairs that have been cataloged 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 to be the job that contains the breakpoint task. (The command used to establish the breakpoint task is contained in the "Breakpoint Commands" section of this chapter.) If there is no breakpoint task, 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 by means of the name .TASK001. If the task is cataloged 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 cataloged, you can use VJ (view job), IJ (inspect job), VT (view task), and IT (inspect task) commands to deduce a token for the task. Then you can define .TASK001 to be a numeric variable whose value is that token.

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

```
──(  D  )──────(    NAME    )────────(  =  )────────(    ITEM    )──────
```

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 an iAPX 86 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

<u>D .TASKA = 2DC3</u>
*

This command creates a symbol called .TASKA in the Debugger's local
symbol table and assigns this symbol the hexadecimal value 2DC3.

LISTING NUMERIC AND BREAKPOINT VARIABLES -- L

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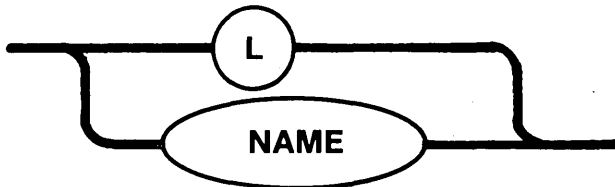
PARAMETER

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 the format:

    NAME=xxxx

where xxxx is the associated value.

EXAMPLES

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

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 four numeric variables (.TASKA, .TASKB, .TASKC, and .TASKD).

LISTING NUMERIC AND BREAKPOINT VARIABLES -- L (continued)

EXAMPLES (continued)

    <u>.TASKA</u>
    TASKA=2DC3
    *

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

CHANGING NUMERIC VARIABLES

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


```
━━━━━(   NAME   )━━━━( = )━━━━(   ITEM   )━━━━
```
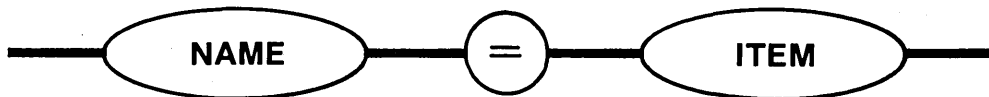

PARAMETERS

    NAME   Name of an existing numeric variable.

    ITEM   An expression or the name of an iAPX 86 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. In a
previous example, .TASKB had a value of 2B8Ch. Therefore, this command
changes the value of .TASKA to 2B8Ch.

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

```
      ( Z )————( NAME )
```

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

    <u>Z .TASKA</u>
    *

This command deletes the numeric variable .TASKA.

BREAKPOINT COMMANDS

The Debugger provides you with the ability to set, change, view, or
delete breakpoints. You set a breakpoint by defining an act which 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 that is 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 things 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
  (are) 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. However, it 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.

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.

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, of the 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, and C for composite).

If the kind 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.

EXCEPTION BREAKPOINT DISPLAY

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 which contains the task that caused the exception condition.
tttt	A token for the task that caused the exceptional conditon.
cccc and iiii	Respectively, the contents of the iAPX 86 CS and IP registers when the exceptional condition occurred.
wwww	The numerical value of the exception code; reflects the nature of the exceptional 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 exceptional condition. If no parameter was at fault, vvvv is 0000.

EXCEPTION BREAKPOINT DIFFERENCES

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

- It is not possible to set, change, view, or delete exception breakpoints by using the commands of the Debugger. Instead, each task can set an exception breakpoint by declaring the Debugger to be its exception handler. The task can subsequently

BREAKPOINT COMMANDS

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) that is used to resume other tasks
on the breakpoint list.

- An exception breakpoint is set for a particular task. Execution
  and exchange breakpoints are set for no particular task; any
  task can incur such a breakpoint.

- An exception breakpoint is not known to the Debugger by a
  breakpoint variable name.

The handling of exception breakpoints is significantly different from
that of execution and exchange breakpoints. For example, exception
breakpoints cannot be viewed, but the other breakpoints can be. Wherever
this distinction applies, this chapter points it out.

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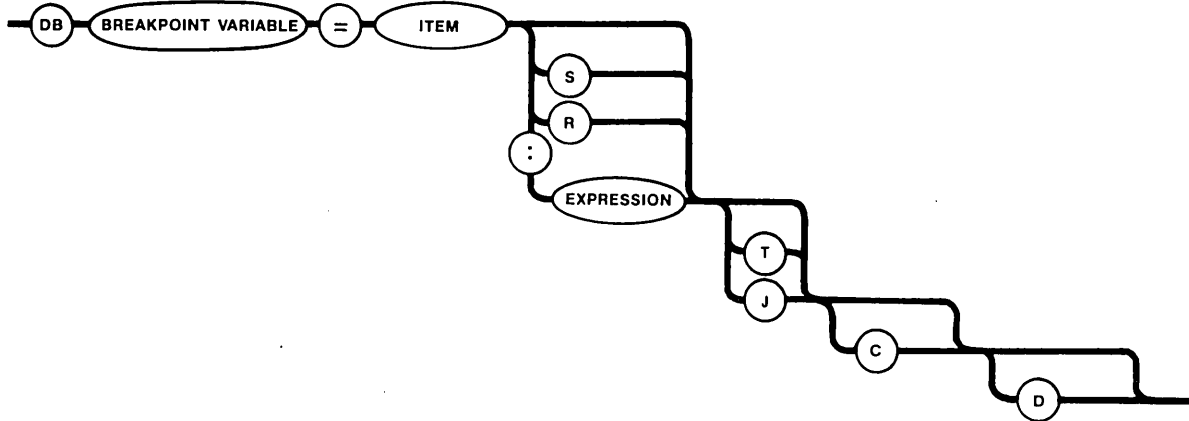
PARAMETERS

BREAKPOINT
VARIABLE
A Debugger name by which to identify 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's display.

ITEM
If you are setting an execution breakpoint, ITEM is used in combination with EXPRESSION to specify the address of the breakpoint. ITEM must contain the base portion of the address. It must be 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 means that it is for 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.

DEFINING A BREAKPOINT -- DB (continued)

PARAMETERS (continued)

EXPRESSION	To be used only when setting an execution breakpoint. EXPRESSION must contain the offset portion of the address of the execution breakpoint.
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. The default is T.
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

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. In a previous example, .MBOX had a value of 2F34; therefore the Debugger uses this value for the token.

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

```
─────( BREAKPOINT
       VARIABLE )─────
```

PARAMETER

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

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).

EXAMINING A BREAKPOINT (continued)

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 variable .BP. It also indicates that only the task is to be
"broken" if a breakpoint is encountered.

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

This command lists the address of the exchange breakpoint associated with
variable .BPOINT. The S, T, and C indicate that only tasks which 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.

CHANGING A BREAKPOINT

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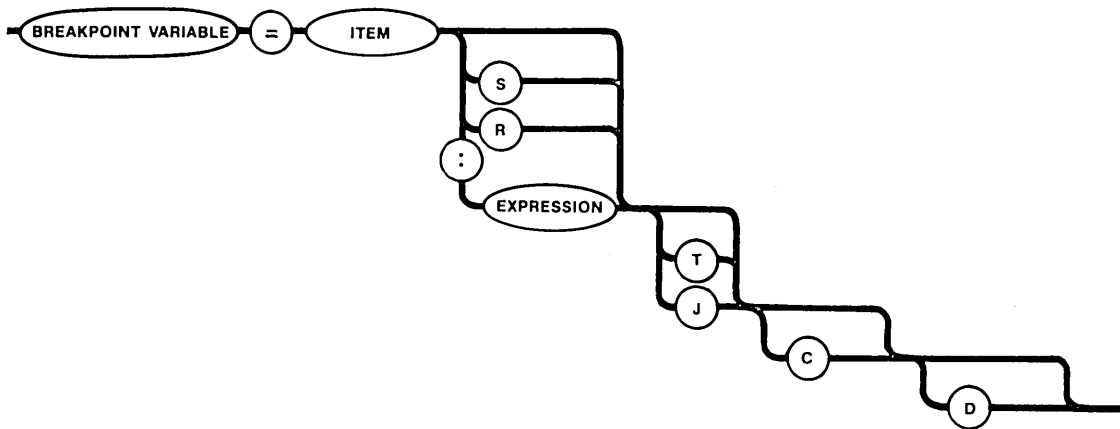
PARAMETERS

BREAKPOINT   An existing Debugger breakpoint name. If the
VARIABLE   Debugger's symbol table does not already contain this
                name, an error message will appear on the terminal's
                display.

ITEM   If you are changing an execution breakpoint, ITEM is
                used in combination with EXPRESSION to specify the
                address of the breakpoint. ITEM must contain the base
                portion of the address. It must be 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 means that it is for 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.

CHANGING A BREAKPOINT (continued)

PARAMETERS (continued)

      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 that was 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.

DELETING A BREAKPOINT -- Z

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

```
    ──────( Z )──────( BREAKPOINT
                       VARIABLE )──────
```
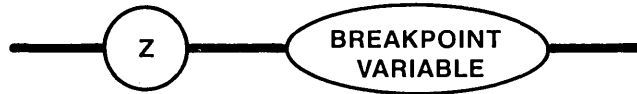
PARAMETER

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

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.

VIEWING THE BREAKPOINT LIST -- BL

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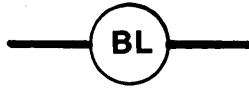
DESCRIPTION

The BL command displays the entire breakpoint list at the user terminal.
This list appears 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).

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

```
——( B )——
```
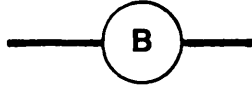
DESCRIPTION

The B command performs the following three functions:

o   Displays the breakpoint list

o   Displays the breakpoint task

o   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).

VIEWING BREAKPOINT PARAMETERS -- B (continued)

DESCRIPTION (continued)

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

The third and final effect of the B command is to display 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.

DESCRIPTION

Breakpoint Variables Display (continued)

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 a 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).

BREAKPOINT COMMANDS

RESUMING TASK EXECUTION -- G

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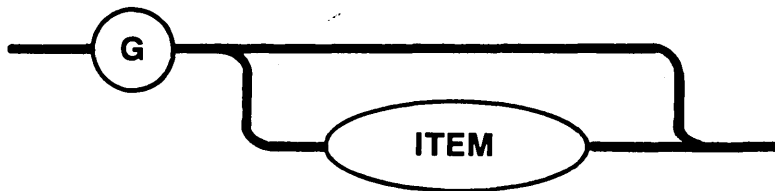
PARAMETER

      ITEM   A token for a task on the breakpoint list or the
breakpoint task. If the given token is not for a task
on the breakpoint list or the breakpoint task, an
error message will be 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 it is 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 at the terminal.

ESTABLISHING THE BREAKPOINT TASK -- BT

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

```
───( BT )───( = )───(    ITEM    )───
```
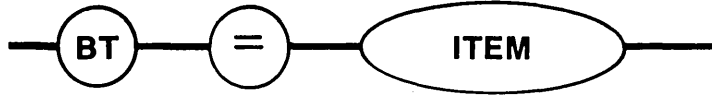
PARAMETER

    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.

LISTING THE BREAKPOINT TASK -- BT

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

```
———( BT )———
```
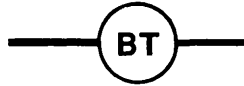
DESCRIPTION

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=0
```

VIEWING THE BREAKPOINT TASK'S REGISTERS -- R

This command lists one or all of the breakpoint task's iAPX 86
registers. The syntax for the R command is as follows:

```
─(R)┬──────────────────────────────────┐
    ├──────────(AH)──────────┤
    ├──────────(AL)──────────┤
    ├──────────(AX)──────────┤
    ├──────────(BH)──────────┤
    ├──────────(BL)──────────┤
    ├──────────(BP)──────────┤
    ├──────────(BX)──────────┤
    ├──────────(CH)──────────┤
    ├──────────(CL)──────────┤
    ├──────────(CS)──────────┤
    ├──────────(CX)──────────┤
    ├──────────(DH)──────────┤
    ├──────────(DI)──────────┤
    ├──────────(DL)──────────┤
    ├──────────(DS)──────────┤
    ├──────────(DX)──────────┤
    ├──────────(ES)──────────┤
    ├──────────(FL)──────────┤
    ├──────────(IP)──────────┤
    ├──────────(SI)──────────┤
    ├──────────(SP)──────────┤
    └──────────(SS)──────────┘
```
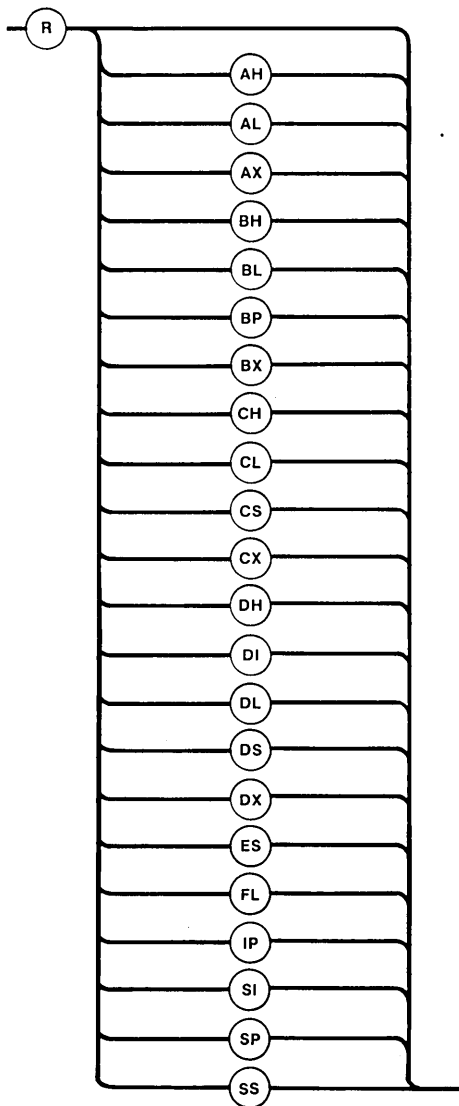
PARAMETERS

    AH, AL, AX,   Names of the breakpoint task's iAPX 86 registers.
    BH, BL, BP,   If no name is specified, the Debugger displays
    BX, CH, CL,   values for all registers.
    CS, CX, DH,
    DI, DL, DS,
    DX, ES, FL,
    IP, SI, SP, SS

BREAKPOINT COMMANDS

VIEWING THE BREAKPOINT TASK'S REGISTERS -- R (continued)

DESCRIPTION

This command lists iAPX 86 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 which prevents the Debugger from obtaining its register contents.
If this is the case, the Debugger displays question marks for all
registers.

ALTERING THE BREAKPOINT TASK'S REGISTERS -- R

This command alters one of the breakpoint task's iAPX 86 register values. The syntax for this command is as follows:
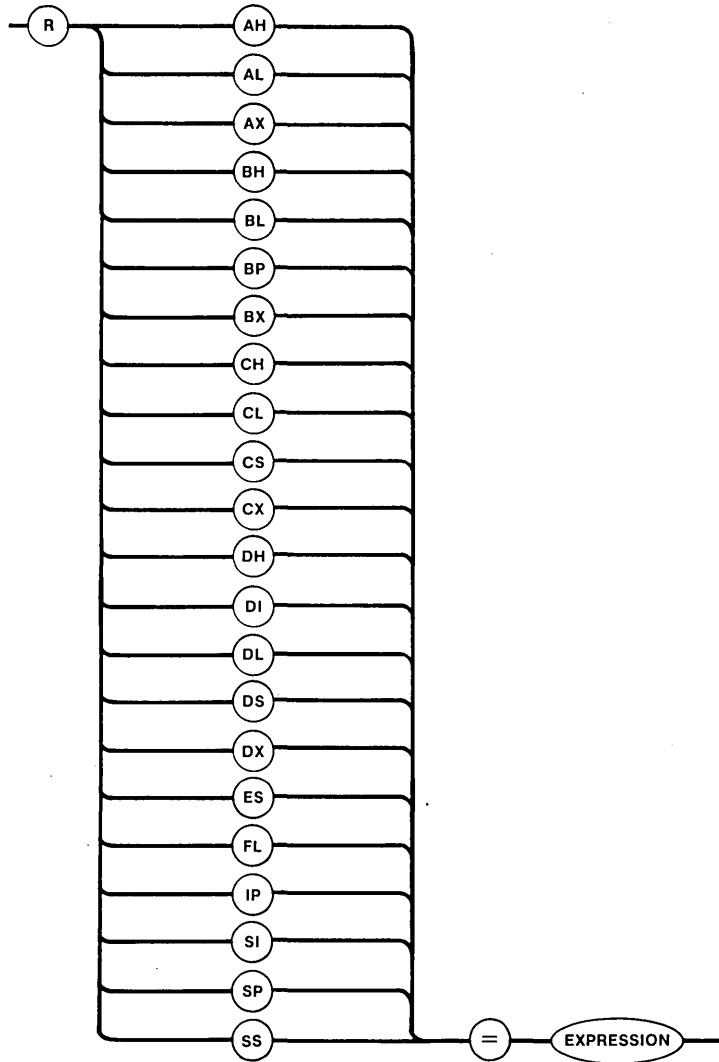
PARAMETERS

AH, AL, AX,   Names of the breakpoint task's iAPX 86 registers.
BH, BL, BP,
BX, CH, CL,
CS, CX, DH,
DI, DL, DS,
DX, ES, FL,
IP, SI,SP, SS

ALTERING THE BREAKPOINT TASK'S REGISTERS -- R (continued)

PARAMETERS (continued)

    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.

VIEWING THE BREAKPOINT TASK'S NDP REGISTERS -- N

This command displays the breakpoint task's 8087 Numeric Data Processor (NDP) register values. This command applies only to tasks that were specified at creation as having the ability to use the 8087 NDP. The syntax for this command is as follows:
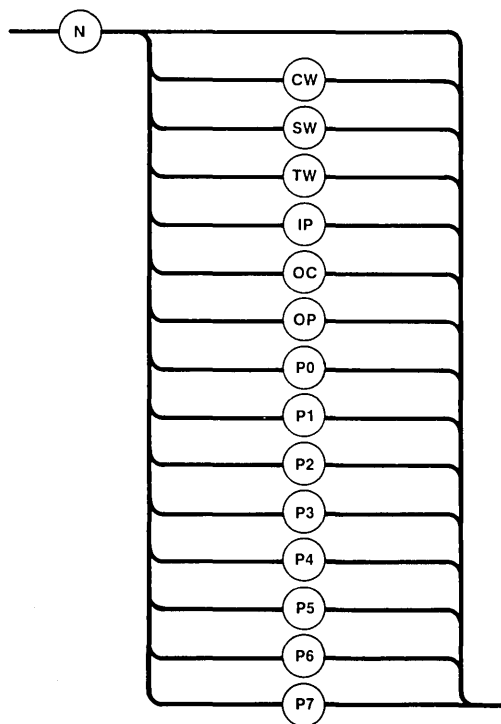
PARAMETERS

CW, SW, TW,   Names of the breakpoint task's 8087 NDP registers,
IP, OC, OP,   as follows:
P0 through P7

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.

VIEWING THE BREAKPOINT TASK'S NDP REGISTERS -- N (continued)

DESCRIPTION

This command lists 8087 NDP register values for the breakpoint task. It
applies only to tasks which were specified at creation as using the 8087
NDP. If the command is simply "N", then all of the breakpoint task's NDP
registers are displayed, in the following format:

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

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. Refer to the 8086 FAMILY USER'S MANUAL,
NUMERICS SUPPLEMENT to translate fields of this format into actual real
numbers.

If the breakpoint task does not use the NDP, the Debugger returns an
error message in response to this command.

ALTERING THE BREAKPOINT TASK'S NDP REGISTERS -- N

This command modifies the breakpoint task's 8087 Numeric Data Processor
(NDP) register values. This command applies only to tasks that were
specified at creation as having the ability to use the 8087 NDP. The
syntax for this command is as follows:
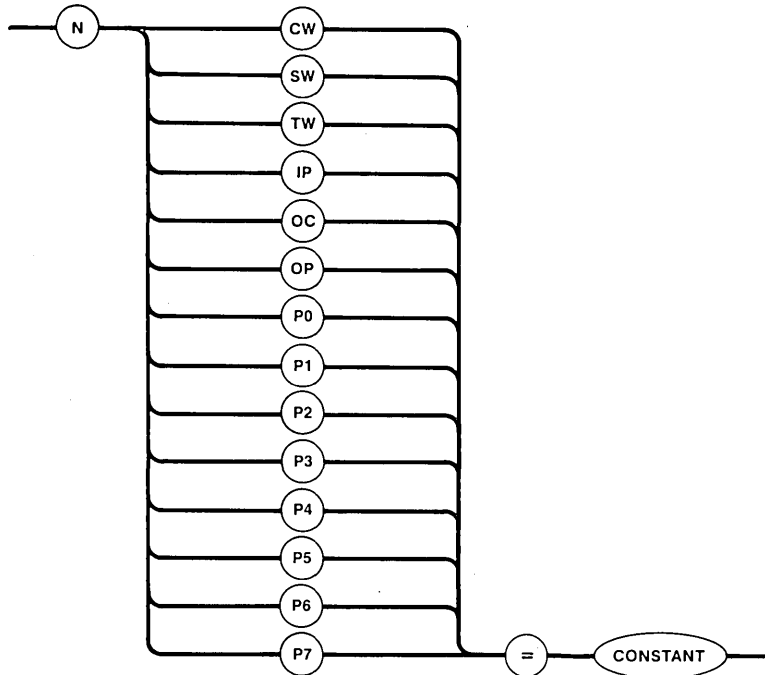
PARAMETERS

CW, SW, TW,   Names of the breakpoint task's 8087 NDP
IP, OC, OP,   registers, as follows:
PO through P7

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

BREAKPOINT COMMANDS

ALTERING THE BREAKPOINT TASK'S NDP REGISTERS -- N (continued)

PARAMETERS (continued)

CONSTANT   A hexadecimal number which is 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 to
                fit in the specified register, the Debugger displays a
                SYNTAX ERROR message.

DESCRIPTION

This command requests that the breakpoint task's NDP register, as
specified in the command request, be updated with the value of CONSTANT.
This command applies only to tasks which were specified at creation as
using the 8087 NDP.

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.
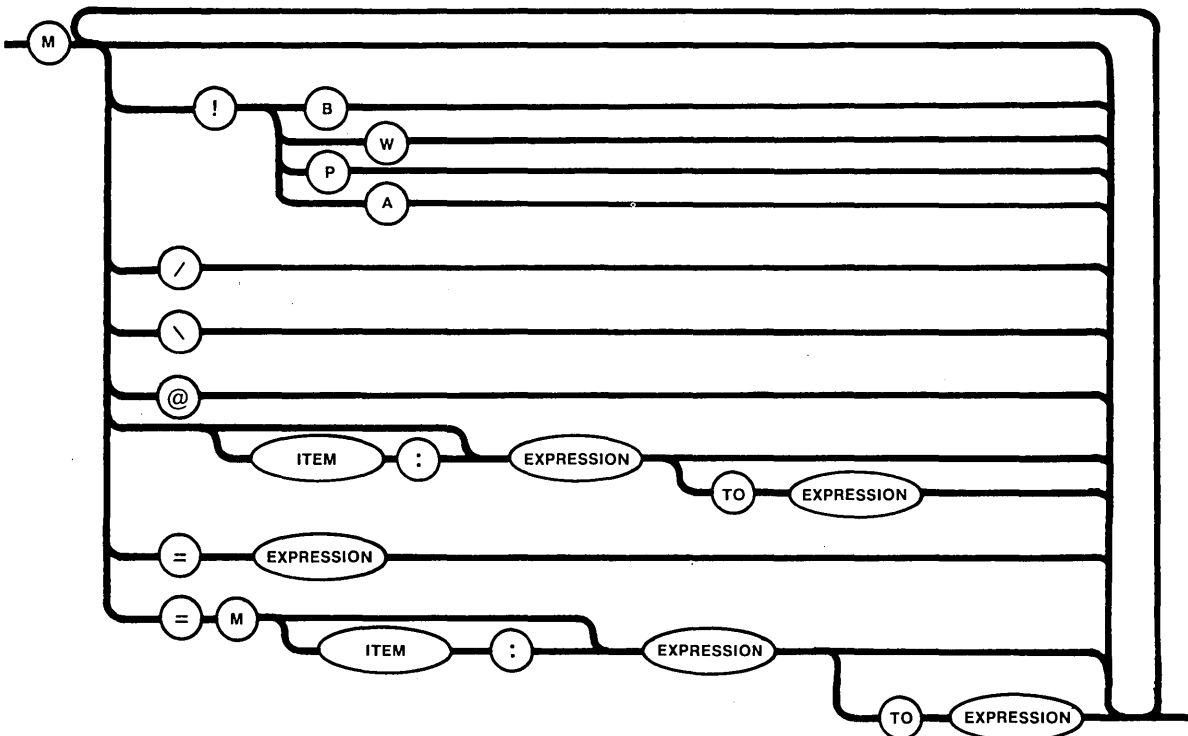
Figure 4-1. Syntax Diagram for Memory Commands

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

o   Setting current display mode. This option begins with "!".

o   Changing memory locations. This option includes the "=".

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

This section breaks up the description of the "M" command into these
three groups and discusses the groups as separate commands. However, you
can combine any number of "M" command options in a single command, as the
syntax diagram in Figure 4-1 illustrates.

In the descriptions of these commands, frequent mention is made of the
current display mode, the current segment base, the current offset, the
current address, and the display of memory locations. This terminology
is defined as follows:

- The current display mode determines the manner in which memory
  values are interpreted for display purposes. The possible modes
  are designated by the letters B, W, P, and A, and they stand,
  respectively, for byte, word, pointer, and ASCII. The effects
  of these modes are best explained in the context of 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 which does hot 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 iAPX 86 segment base. The initial value of
  the current segment base is 0.

- The current offset is a value the Debugger maintains and uses
  when reference is made to a memory location without explicitly
  citing an 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 iAPX 86 memory address computed from
  the combination of the current segment base and the current
  offset.

4-38

● 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 in a single request, 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 represent 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 of them begins at an offset which is a multiple of 16 (10 hexadecimal).

MEMORY COMMANDS

SETTING THE CURRENT DISPLAY MODE -- M

This command specifies the way in which the Debugger will display
output. The syntax for the M command is as follows:
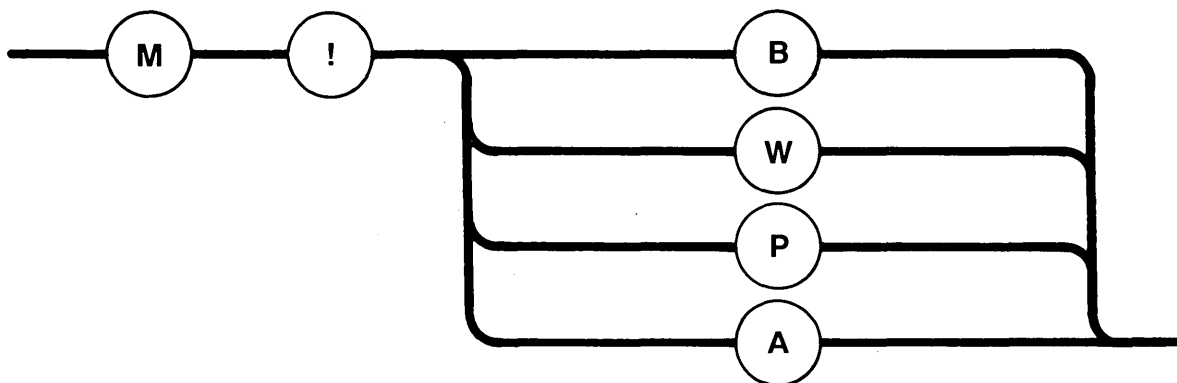
PARAMETERS

>   !   Indicates that the display mode is being changed.

>   B, W,   Specifies the mode of display. B indicates byte mode,
>   P, A   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
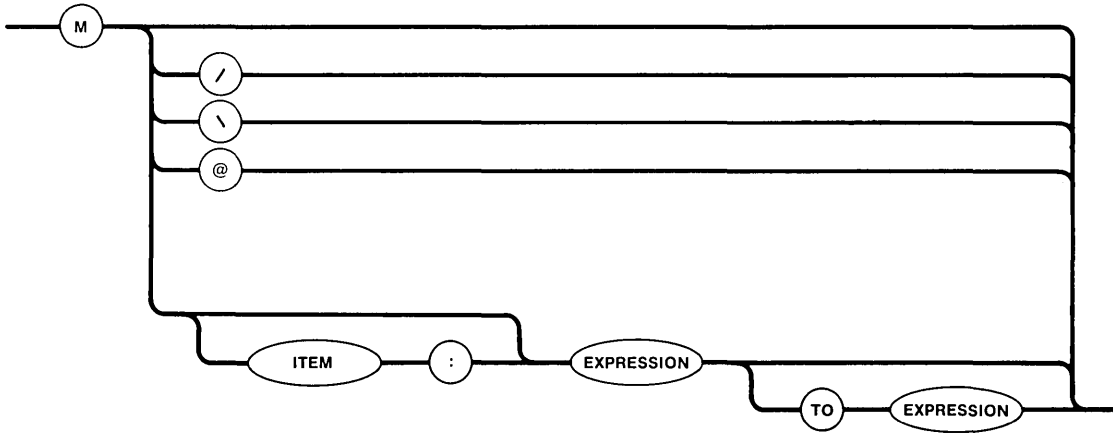the mode specified with this command.

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.

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.

EXAMINING MEMORY -- M (continued)

PARAMETERS (continued)

M   When used by itself, M is an abbreviated way of
             specifying M/ or M\, whichever was used most
             recently. 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:EX-   This option is just like M EXPRESSION, except that
  PRESSION   ITEM is used as the base in the address calculation,
             and after the operation ITEM is the new current
             segment base.

PARAMETERS

M ITEM:EXPRESSION (continued)

> 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 This option displays the values of a series of
TO EXPRESSION 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:EX-   This option is just like M EXPRESSION TO EXPRESSION,
PRESSION TO   except that ITEM is used as a base in the address
EXPRESSION   calculation, and after the operation ITEM is the new
segment base.

EXAMINING MEMORY -- M (continued)

PARAMETERS

M ITEM:EXPRESSION TO EXPRESSION (continued)

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). Upon completion of
          this 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
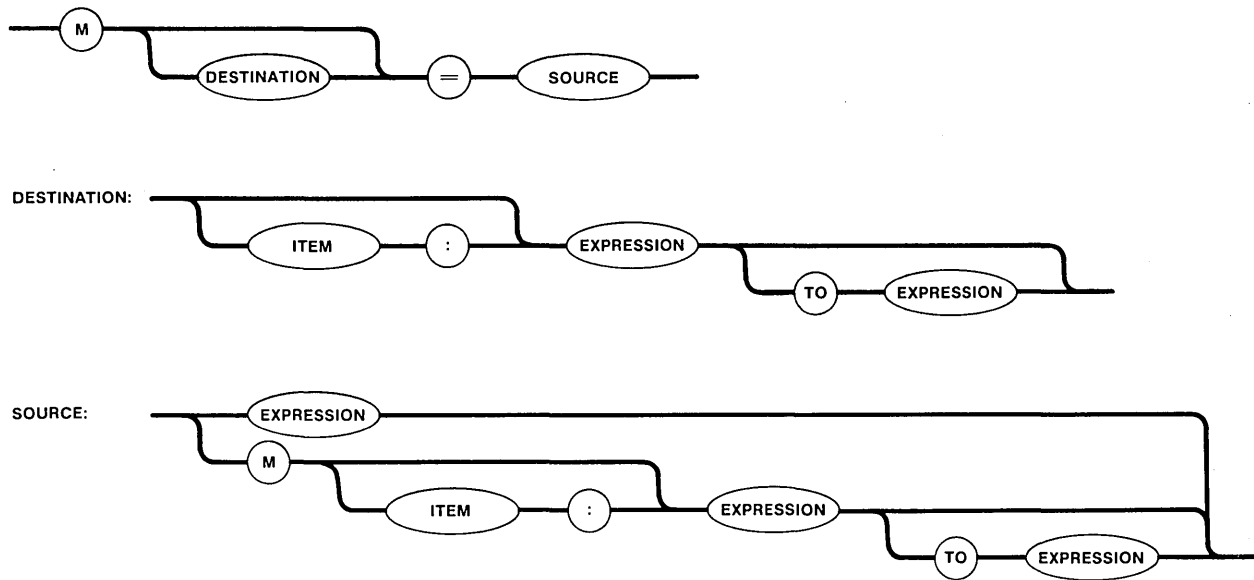the current display mode.

This command changes the contents of designated RAM locations.

CAUTION

Because the Debugger is generally used
during system development, while your
tasks, the Nucleus, the Debugger, and
possibly other iRMX 86 components are
in RAM, you should use these M command
options with extreme care.

The syntax for this command is as follows:

MEMORY COMMANDS

PARAMETERS

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

CHANGING MEMORY -- M (continued)

PARAMETERS (continued)

Destination Parameters

These parameters define the memory location or locations that are going
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:EX-   This form of the DESTINATION option implies that the
  PRESSION   address to be changed has the value of ITEM as its
   base value and the value of EXPRESSION as its offset.

EXPRESSION TO   This form of the DESTINATION option implies that a
  EXPRESSION   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:EX-   This form of the DESTINATION option is the same as the
  PRESSION TO   previous one, except that ITEM is used as the base
  EXPRESSION   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.
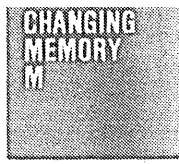
PARAMETERS (continued)

Source Parameters

These parameters define the information that will 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:
    <u>M = 4C</u>
    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:
    <u>M 1 TO 4</u>
    0400:0001 06 07 08 09
    *
    <u>M = 4C</u>
    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.

CHANGING MEMORY -- M (continued)

PARAMETERS

Source Parameters

M EXPRESSION (continued)

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

M ITEM:EX- This form of the SOURCE option uses ITEM and
  PRESSION 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 of the locations in the destination field.

PARAMETERS

Source Parameters

M ITEM:EXPRESSION (continued)

> 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 of the locations
> 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.

CHANGING MEMORY -- M (continued)

PARAMETERS

Source Parameters

M EXPRESSION TO EXPRESSION (continued)

Examples:

(1) When the DESTINATION option did not specify a range of values:
```
    M 400:104
    0400:0104 El
    *
    M = M A TO C
    0400:0104 El 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:EX-
PRESSION 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.

PARAMETERS

Source Parameters

M ITEM:EXPRESSION TO EXPRESSION (continued)

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.

CHANGING MEMORY -- M (continued)

DESCRIPTION (continued)

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, the information you need to restore it is available.

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

NOTE

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

● It is possible for you to modify memory within iRMX 86 components, such as the Nucleus and Debugger. Doing so can jeopardize the integrity of your application system, and should therefore be avoided.

● It is possible to 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.

COMMANDS TO INSPECT SYSTEM OBJECTS

The commands in this section allow you to examine iRMX 86 objects in detail. They give specific information about the Nucleus object types. Figure 4-2 illustrates the general syntax for all the commands in this section.
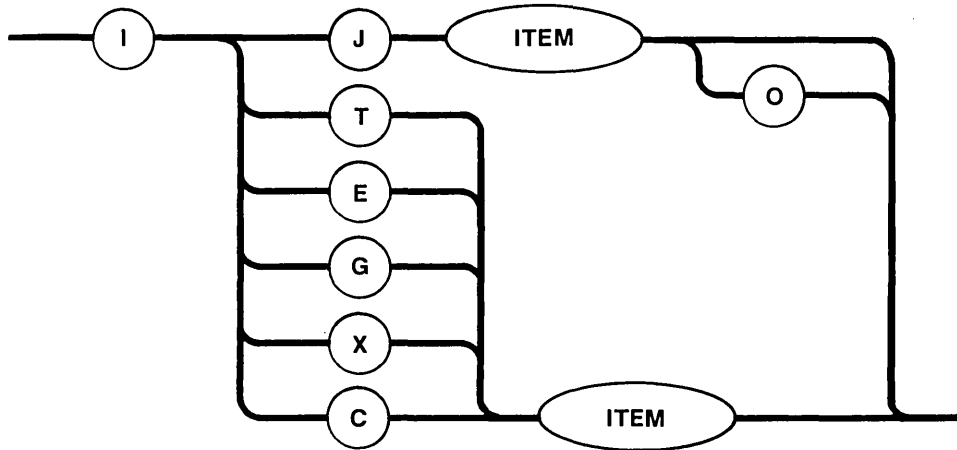

Figure 4-2. Syntax Diagram for Inspecting System Objects

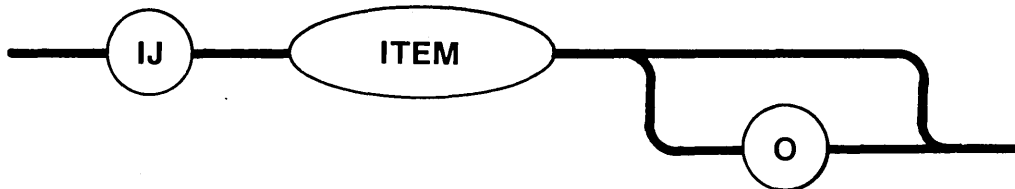The second letter of the command indicates the type of object to inspect, as follows:

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

The remainder of this section describes the commands in detail.

INSPECTING A JOB -- IJ

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

```
 ──────────( IJ )────────(      ITEM      )─────────┐          ┌────
                                                    │  ┌───┐   │
                                                    └──( O )───┘
```

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 there is a large number of entries 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 4-3 depicts the form of the display produced by the IJ command.
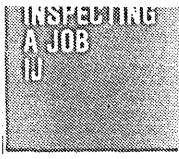
DESCRIPTION (continued)

```
                      ----- iRMX86 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
        ssssssssssss tttt      ssssssssssss tttt     ssssssssssss tttt
```

Figure 4-3. An iRMX 86™ Job Report

The following describes the fields in Figure 4-3:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the job is cataloged in the object directory of either the job's parent job or the root job. If the job is not cataloged in either directory, "NONE FOUND" is printed 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 that have been 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.

INSPECTING A JOB -- IJ (continued)

DESCRIPTION (continued)

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

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

Field	Meaning
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.
1111	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 (which is described in the iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL). 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.

DESCRIPTION (continued)

Field	Meaning
ssssssssssss	Each such field contains the name under which an object is cataloged in the job's object directory. If there are no entries 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.

INSPECT COMMANDS

INSPECTING A TASK -- IT

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

IT ── ITEM

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 4-4 depicts the form of display produced
by IT.

```
                     ----- iRMX86 TASK REPORT -----
          TASK TOKEN              bbbb      CONTAINING JOB              kkkk
          STACK SEGMENT BASE      cccc      STACK SEGMENT OFFSET        1111
          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
          NDP TASK               jjj
          NAME(S)  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
                   aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa  aaaaaaaaaaaa
```

Figure 4-4. An iRMX 86™ Task Report

The following describes the fields in Figure 4-4:

Field   Meaning

aaaaaaaaaaaa   Each such field contains a name under which the task
                    is cataloged in the object directory of either the
                    task's containing job or the root job. If the job is
                    not cataloged in either directory, "NONE FOUND" is
                    displayed here.

DESCRIPTION (continued)

Field	Meaning
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

Field	Meaning
jjj	Indicator which tells whether the task uses the 8087 NDP. The possible values of jjj 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", and "BROKEN".

INSPECT COMMANDS

INSPECTING A TASK -- IT (continued)

DESCRIPTION (continued)

Field	Meaning
qqqq	A temporary, hexadecimal priority that is sometimes assigned to the task by the Nucleus. This is done to improve system performance.
rrrr	If the task is asleep or asleep/suspended, this is the number of 1/100 second sleep units that the task requested just prior to 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 a mailbox, semaphore,
or region whose token is specified. The syntax of the IE command is as
follows:

```
    ──( IE )──────( ITEM )──────
```

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

```
                    ───── iRMX86 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
```
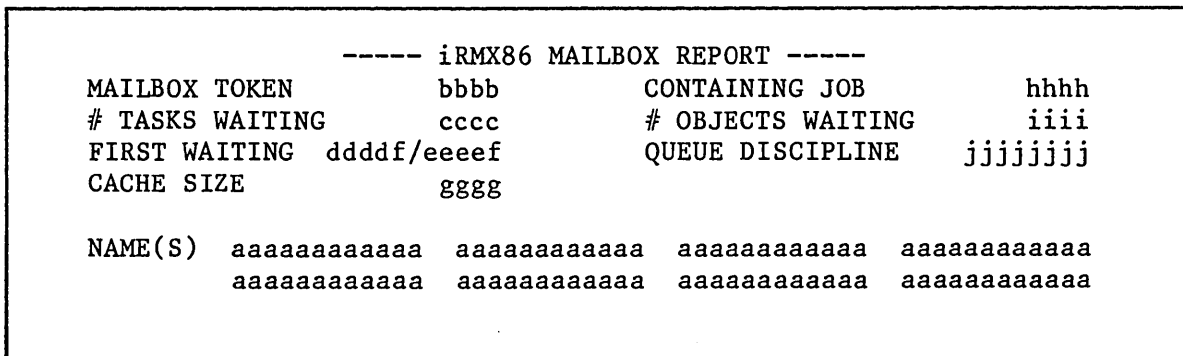
Figure 4-5. An iRMX 86™ Mailbox Report

INSPECT COMMANDS

INSPECTING AN EXCHANGE -- IE (continued)

DESCRIPTION

Mailbox Display (continued)

The following describes the fields in Figure 4-5:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the mailbox is cataloged in the object directory of either the mailbox's containing job or the root job. If the mailbox is not cataloged 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 blank. Otherwise, f has one of the following values:

> .C composite
> G segment
> J job
> M mailbox
> R region
> S semaphore
> T task
> X extension

DESCRIPTION

Mailbox Display (continued)

Field	Meaning
gggg	Number, in hexadecimal, of objects that the mailbox's high performance object queue is capable of holding.
hhhh	Hexadecimal token for the job containing the mailbox.
iiii	Number, in hexadecimal, of objects in the mailbox's object queue.
jjjjjjjj	Description of the manner in which waiting tasks are queued in the mailbox's task queue. The possible values are FIFO and PRIORITY.

Semaphore Display

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

```
                ----- iRMX86 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 4-6. An iRMX 86™ Semaphore Report

The following describes the fields in Figure 4-6:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the semaphore is cataloged in the object directory of either the semaphore's containing job or the root job. If the semaphore is not cataloged in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the semaphore.

INSPECTING AN EXCHANGE -- IE (continued)

Semaphore Display (continued)

Field	Meaning
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	Description of the manner in which waiting tasks are queued in the semaphore's task queue. The possible values are FIFO and PRIORITY.
iiii	Maximum allowable number, in hexadecimal, of units that the semaphore may have in its custody.

Region Display

Figure 4-7 depicts the form of display produced by IE for a region.

```
-------------------------------------------------------------------------
                   ----- iRMX86 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 4-7. An iRMX 86™ Region Report

DESCRIPTION
Region Display (continued)

The following describes the fields in Figure 4-7:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the region is cataloged in the object directory of either the job containing the region or the root job. If the region is not cataloged 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	Manner in which waiting tasks are queued at the region. Possible values are FIFO and PRIORITY.

INSPECTING A SEGMENT -- IG

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

```
      ──( IG )──( ITEM )──
```

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 4-8 depicts the form of the display
produced by IG.

```
                  ───── iRMX86 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 4-8. An iRMX 86™ Segment Report

The following describes the fields in Figure 4-8:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the segment is cataloged in the object directory of either the segment's containing job or the root job. If the segment is not cataloged in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the segment.
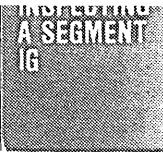
DESCRIPTION (continued)

Field	Meaning
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.

INSPECT COMMANDS

INSPECTING AN EXTENSION -- IX

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

```
 ──( IX )──( ITEM )──
```

PARAMETER

    ITEM   Token for the extension object to be inspected.

DESCRIPTION

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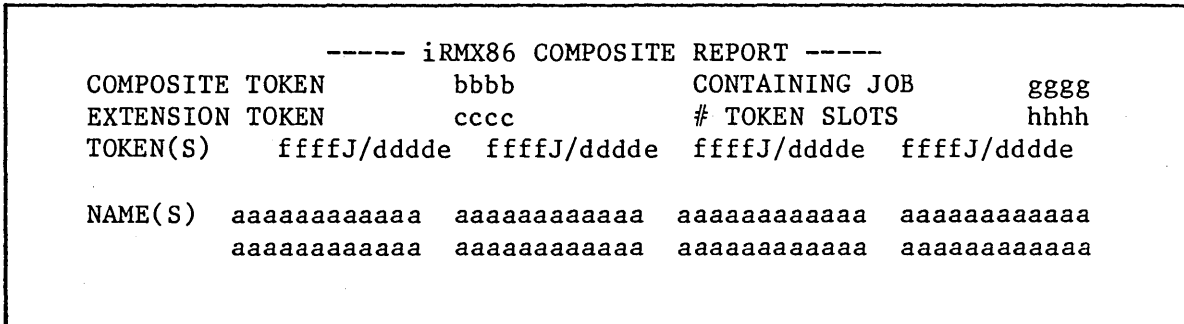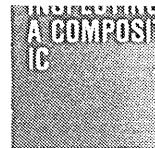token is represented by ITEM. Figure 4-9 depicts the form of the display
produced by IX.

```
        ----- iRMX86 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 4-9. An iRMX 86™ Extension Report

The following describes the fields in Figure 4-9:

Field	Meaning
aaaaaaaaaaaa	Each such field contains a name under which the extension is cataloged in the object directory of either the job containing the extension or the root job. If the extension is not cataloged in either directory, "NONE FOUND" is displayed here.
bbbb	Hexadecimal token for the extension.

DESCRIPTION (continued)

Field	Meaning
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.

INSPECTING A COMPOSITE -- IC

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

```
──( IC )──( ITEM )──
```

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. Figure 4-10 depicts the form of the display produced by IC.

```
                ----- iRMX86 COMPOSITE REPORT -----
    COMPOSITE TOKEN        bbbb          CONTAINING JOB        gggg
    EXTENSION TOKEN        cccc          # TOKEN SLOTS         hhhh
    TOKEN(S)    ffffJ/ddde  ffffJ/ddde  ffffJ/ddde  ffffJ/ddde


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

Figure 4-10. An iRMX 86™ Composite Report

The following describes the fields in Figure 4-10:

  Field   Meaning

aaaaaaaaaaaa   Each such field contains a name under which the
                      composite is cataloged in the object directory of
                      either the job containing the composite or the root
                      job. If the composite is not cataloged in either
                      directory, "NONE FOUND" is displayed here.

  bbbb   Hexadecimal token for the composite.

DESCRIPTION (continued)

Field	Meaning
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 86 Application Loader

Field	Meaning
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.

COMMANDS TO VIEW OBJECT LISTS

The commands in this section allow you to view lists of iRMX 86 objects.
Figure 4-11 illustrates the general syntax for commands in this section.
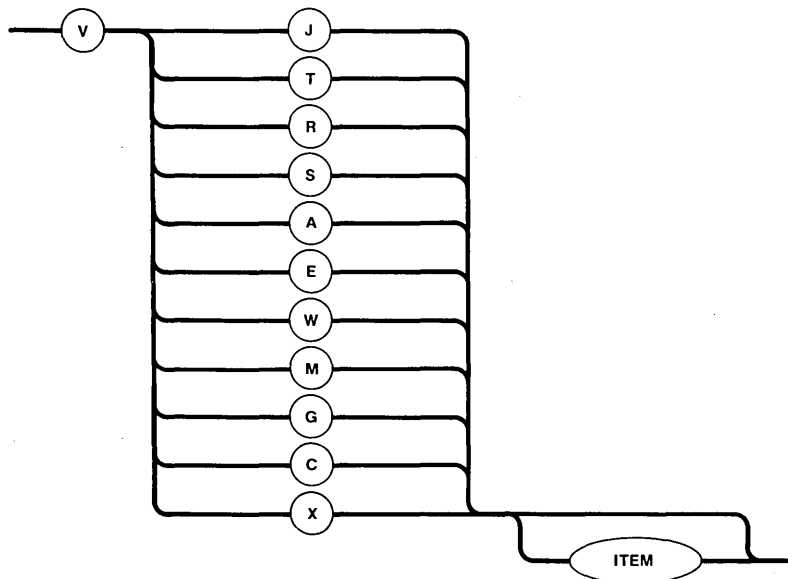
Figure 4-11. Syntax Diagram for Viewing iRMX 86™ Object Lists

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

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

The remainder of this section describes the commands in detail.

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

```
        ┌──(VJ)──┬──────────────────┬──
                 │    ┌─────────┐    │
                 └────(  ITEM   )────┘
                      └─────────┘
```
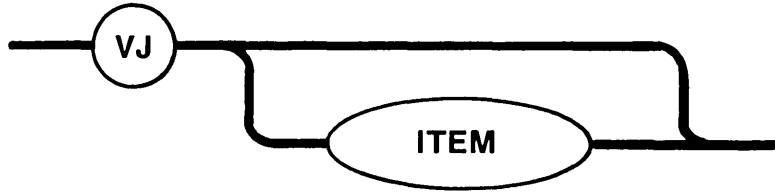
PARAMETER

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".

VIEW COMMANDS

VIEWING TASKS -- VT

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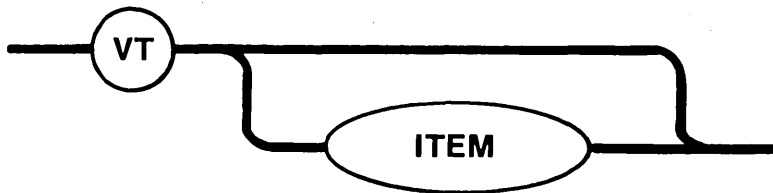
PARAMETER

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

DESCRIPTION

The VT command displays tasks as:
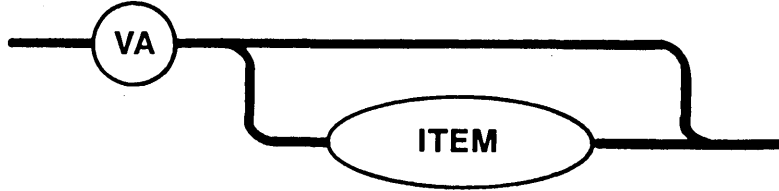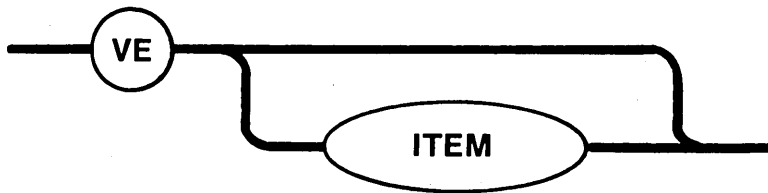
```
TL = jjjjJ/ttttT jjjjJ/ttttT ... jjjjJ/ttttT
```

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.

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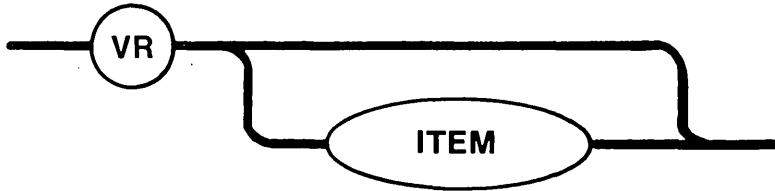
PARAMETER

ITEM   Token for a job. If this option is included, the
                         Debugger lists, in priority order, the ready tasks
                         that are 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:
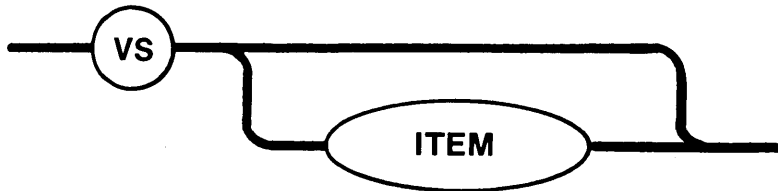
PARAMETER

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 asleep tasks. The syntax for the VA
command is as follows:

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 suspended 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.

VIEWING EXCHANGES -- VE

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

```
       ┌────┐
───────( VE )──────┬──────────────────────────┬────
       └────┘      │      ╭───────────╮        │
                   └──────( ITEM )─────────────┘
                          ╰───────────╯
```

PARAMETER

    ITEM   Token for a job. If this option is included, the
Debugger lists only those exchanges that are 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 the waiting task queues at exchanges. The syntax for the VW command is as follows:

```
  ──(VW)──┬─────────────────┬──
          │                 │
          └──( ITEM )──┘
```
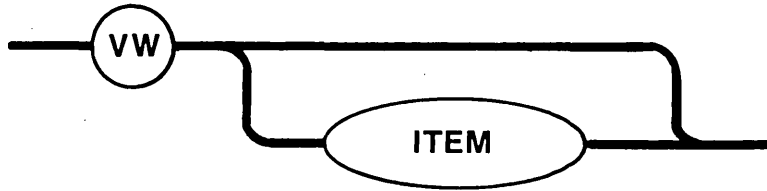
PARAMETER

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,
                or R for region).

tttt   Token for a task which is 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.

VIEW COMMANDS

VIEWING MAILBOX OBJECT QUEUES -- VM

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

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 segments. The syntax for the VG command
is as follows:

```
  ──────○─( VG )────────────────────────○┐
                  └──────(    ITEM    )──────┘
```
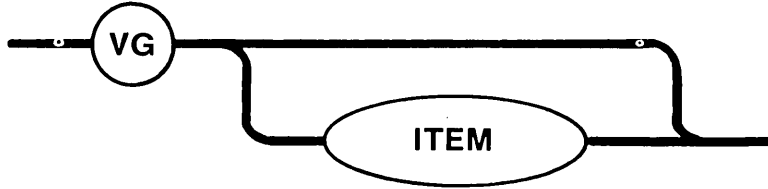
PARAMETER

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 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:
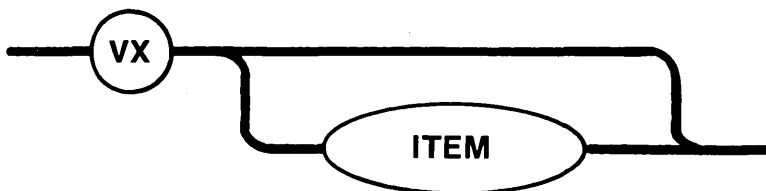
PARAMETER

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.

EXTENSIONS
VX

DESCRIPTION (continued)

jjjj   Token for the job containing the extension.

cccc   Token for the composite object that is associated with
            the specified extension.

kkkk   Token for the job containing the composite object.

VIEW COMMANDS

VIEWING COMPOSITES -- VC

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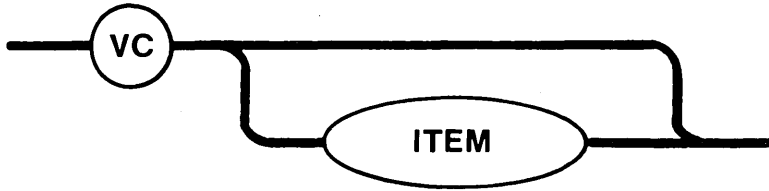
PARAMETER

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.

COMMANDS TO EXIT THE DEBUGGER

The Q command described in this section allows you to exit the Debugger
and resume processing.

EXIT COMMANDS

EXITING THE DEBUGGER -- Q

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

$$\longrightarrow\!\!\!\bigcirc\!\!\!\!\!\text{Q}\!\!\!\!\!\longrightarrow$$

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 86 DEBUGGER".

APPENDIX A. 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:

    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 NDP TASK	You specified the N command, but the breakpoint task was not designated as an 8087 task at its creation.
COMMAND TOO COMPLEX	In order to process your commands, the Debugger maintains a semantic stack, on which it places all the semantic entities of your command. Your command was too complex and overflowed this stack. To correct this problem, you should first define numeric variables for some of your more complex expressions, and then use these variables in your command in place of the expressions.
DEBUGGER POOL TOO SMALL	In order to process your command, the Debugger tried to create an iRMX 86 segment. However, there was not enough free space 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
INVALID TASK STATE	The Nucleus—maintained task descriptor contains inconsistent information. You have probably overwritten this area of memory. It is unlikely that the task can continue to run.
INVALID TOKEN	You specified a token for a different kind of object than that required by the command.
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 iSBC861 NOT CONFIGURED	The Debugger encountered a breakpoint for which it had no record. It tried to pass the breakpoint to the iSBC 957A monitor but could not because the iSBC 957A package is not included in your system.

REQUEST FOR READER'S COMMENTS

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

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

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

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

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

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____
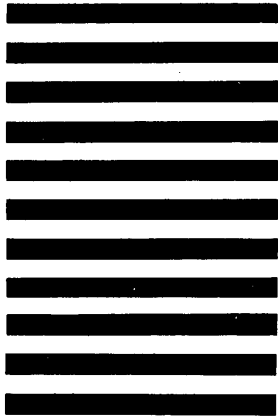
Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS . . .

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

intel®