

**iRMX 86™ NUCLEUS, TERMINAL
HANDLER, AND DEBUGGER
REFERENCE MANUAL**

Manual Number 9803122-02

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

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

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

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

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

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

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

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

BXP
CREDIT
i
iCE
iCS
im
Insite
Intel

Intel
Intelelevision
Intellec
iRMX
iSBC
iSBX
Library Manager
MCS

Megachassis
Micromap
Multibus
MULTIMODULE
PROMPT
Promware
RMX/80
UPI
 μ Scope

and the combination of ICE, iCS, iRMX, iSBC, iSBX, or MCS and a numerical suffix.

PREFACE

iRMX 86 provides an operating system for Intel's iSBC 86/12 single board computers. It consists of a Nucleus, a Terminal Handler, a Debugger, and an input/output system (IOS).

The Nucleus, Terminal Handler, and Debugger Reference Manual is one of eight manuals supporting the iRMX 86 Operating System. The other manuals are:

<u>Introduction to the iRMX 86 Operating System</u>	9803124
<u>iRMX 86 Installation Guide for ISIS-II Users</u>	9803125
<u>iRMX 86 I/O System Reference Manual</u>	9803123
<u>iRMX 86 System Programmer's Reference Manual</u>	142721
<u>iRMX 86 Configuration Guide for ISIS-II Users</u>	9803126
<u>Guide to Writing Device Drivers for the iRMX 86 I/O System</u>	142926
<u>iRMX 86 Programming Techniques</u>	142982

This manual is intended primarily as a source of Nucleus, Terminal Handler, and Debugger reference materials; it is only secondarily for instruction. We recommend reading the introductory manual prior to reading this manual.

The iRMX 86 manual set is aimed at two classes of readers: application programmers and system programmers. Accordingly, reference information is separated by class. In particular, this manual and the I/O system manual are for application programmers, while systems-oriented features in both the Nucleus and the I/O System are described in the System Programmer's manual.

The following manuals provide valuable background information:

<u>iSBC 86/12A Hardware Reference Manual</u>	9803074
<u>ISIS-II User's Guide</u>	9800306
<u>PL/M-86 Programming Manual</u>	9800466
<u>ISIS-II PL/M-86 Compiler Operator's Manual</u>	9800478
<u>The 8086 Family User's Manual</u>	9800722

CONTENTS

	PAGE
PREFACE	iii
CHAPTER 1	
OVERVIEW	
Nucleus.....	1-1
Terminal Handler.....	1-1
Debugger.....	1-1
CHAPTER 2	
NUCLEUS OVERVIEW	
Introduction.....	2-1
Objects.....	2-2
Tasks.....	2-2
Jobs.....	2-3
Segments.....	2-4
Mailboxes.....	2-4
Semaphores.....	2-4
Handlers.....	2-5
Exception Handlers.....	2-5
Interrupt Handlers.....	2-5
CHAPTER 3	
JOB MANAGEMENT	
Job Tree and Resource Sharing.....	3-1
Job Creation.....	3-3
Job Deletion.....	3-3
System Calls For Jobs.....	3-3
CHAPTER 4	
TASK MANAGEMENT	
Task States.....	4-1
The Asleep State.....	4-1
The Suspended State.....	4-1
The Asleep-Suspended State.....	4-1
The Ready and Running State.....	4-2
Task State Transitions.....	4-2
Additional Task Attributes.....	4-4
Task Resources.....	4-4
System Calls for Tasks.....	4-4
CHAPTER 5	
EXCHANGE MANAGEMENT	
Mailboxes.....	5-1
Mailbox Queues.....	5-1
Mailbox Mechanics.....	5-1
System Calls for Mailboxes.....	5-2
Semaphores.....	5-3
Semaphore Queue.....	5-3
Semaphore Mechanics.....	5-3
System Calls for Sempahores.....	5-4

CONTENTS (continued)

	PAGE
CHAPTER 6	
MEMORY MANAGEMENT	
Segments.....	6-1
Memory Pools.....	6-1
Controlling Pool Size.....	6-2
Movement of Memory Between Jobs.....	6-2
Memory Allocation.....	6-3
System Calls for Segments.....	6-4
CHAPTER 7	
OBJECT MANAGEMENT.....	7-1
System Calls for Any Objects.....	7-2
CHAPTER 8	
EXCEPTIONAL CONDITION MANAGEMENT	
Types of Exceptional Conditions.....	8-1
Exception Handlers.....	8-1
Assigning An Exception Handler.....	8-2
Invoking An Exception Handler.....	8-2
Handling Exceptions In-Line.....	8-4
System Calls For Exception Handlers.....	8-4
CHAPTER 9	
INTERRUPT MANAGEMENT	
Interrupt Mechanisms.....	9-1
The Interrupt Vector Table.....	9-1
Interrupt Levels.....	9-1
Disabling Interrupts.....	9-2
Interrupt Handlers and Interrupt Tasks.....	9-3
Setting Up An Interrupt Handler.....	9-3
Using an Interrupt Handler.....	9-4
Using and Interrupt Task.....	9-4
Handling Level 7 Interrupts.....	9-7
Examples of Interrupt Servicing.....	9-7
System Calls for Interrupts.....	9-10
CHAPTER 10	
NUCLEUS SYSTEM CALLS.....	10-1
Command Directory.....	10-2
Calls for Jobs.....	10-2
Calls for Tasks.....	10-2
Calls for Mailboxes.....	10-2
Calls for Semaphores.....	10-3
Calls for Segments and Memory Pools.....	10-3
Calls for All Objects.....	10-3
Calls for Exception Handlers.....	10-4
Calls for Interrupt Handlers, Tasks, and Levels.....	10-4
The System Calls.....	10-5
Catalog Object.....	10-5
Create Job.....	10-7
Create Mailbox.....	10-13
Create Segment.....	10-14
Create Semaphore.....	10-15
Create Task.....	10-17

CONTENTS (continued)

	PAGE
CHAPTER 10 (continued)	
Delete Job.....	10-20
Delete Mailbox.....	10-22
Delete Segment.....	10-23
Delete Semaphore.....	10-24
Delete Task.....	10-25
Disable.....	10-26
Enable.....	10-27
Enter Interrupt.....	10-28
Exit Interrupt.....	10-30
Get Exception Handler.....	10-31
Get Level.....	10-33
Get Pool Attrib.....	10-34
Get Priority.....	10-36
Get Size.....	10-37
Get Task Tokens.....	10-38
Get Type.....	10-39
Lookup Object.....	10-40
Offspring.....	10-42
Receive Message.....	10-43
Receive Units.....	10-45
Reset Interrupt.....	10-47
Resume Task.....	10-48
Send Message.....	10-49
Send Units.....	10-51
Set Exception Handler.....	10-52
Set Interrupt.....	10-54
Set Pool Minimum.....	10-58
Signal Interrupt.....	10-59
Sleep.....	10-60
Suspend Task.....	10-61
Uncatalog Object.....	10-62
Wait Interrupt.....	10-63
 CHAPTER 11	
TERMINAL HANDLER.....	11-1
General Information.....	11-1
Using A Terminal With the iRMX 86 Operating System.....	11-1
How Normal Characters are Handled.....	11-2
How Special Characters are Handled.....	11-2
Rubbing Out a Previously-Typed Character.....	11-3
Displaying the Current Line.....	11-3
Deleting the Current Line.....	11-3
Sending an Empty Message.....	11-4
Signalling the End of a Line of Input.....	11-4
Output Control.....	11-4
Suspending Output.....	11-4
Resuming Output.....	11-5
Deleting or Restarting Output.....	11-5
Program Control.....	11-5
Calling a User-written Procedure Manually.....	11-5
Setting A Baud Rate.....	11-5
Programming Considerations.....	11-6
Output.....	11-7
Input.....	11-8

CONTENTS (continued)

	PAGE
CHAPTER 12	
DEBUGGER	
General Information.....	12-1
Debugger Capabilities.....	12-1
Debugging Capabilities in the ICE-86 Emulator.....	12-2
Debugging Capabilities in the iSBC 86/12A Monitor.....	12-2
Invoking the Debugger.....	12-2
Debugger Input and Output.....	12-3
Syntax of Debugger Commands.....	12-4
Pictorial Representation of Syntax.....	12-4
Special Symbols for the Debugger.....	12-5
Debugger Commands.....	12-8
Breakpoint Control.....	12-8
Establishing a Breakpoint.....	12-11
Changing a Breakpoint.....	12-13
Deleting a Breakpoint.....	12-14
Examining a Breakpoint.....	12-15
Viewing the Breakpoint List.....	12-16
Viewing the Breakpoint Parameters.....	12-17
Removing a Task from the Breakpoint List.....	12-19
Establishing the Breakpoint Task.....	12-19
Inquiring as to the Breakpoint Task.....	12-20
Viewing the Breakpoint Task's Registers.....	12-21
Altering the Breakpoint Task's Registers.....	12-23
Observation and Manipulation Commands.....	12-24
Examining or Modifying Memory.....	12-25
Examining System Objects.....	12-33
Viewing iRMX 86 System Lists.....	12-40
Exiting the Debugger.....	12-44
Using Symbolic Names While Debugging.....	12-44
Defining a Numeric Variable.....	12-45
Changing a Numeric Variable.....	12-45
Deleting a Numeric Variable.....	12-46
Viewing Numeric Variables.....	12-46
APPENDIX A	
iRMX 86 Data Types.....	A-1
APPENDIX B	
iRMX 86 Type Codes.....	B-1

TABLES

		PAGE
8-1	Conditions and their Codes.....	8-3
9-1	Interrupt Levels Disabled For Running Task.....	9-2
9-2	The Relationship between External Levels and Internal Task Priorities.....	9-7
9-3	An Example of Interrupt Handling Without an Interrupt Task.....	9-8
9-4	An Example of Interrupt Handling with an Interrupt Task.....	9-9
11-1	Special Character Summary.....	11-2

ILLUSTRATIONS

3-1	A Job.....	3-2
4-1	Task State Transition Diagram.....	4-3
6-1	Comparison of Job and Memory Hierarchies.....	6-2
6-2	Memory Movement Diagram.....	6-3
9-1	Flow Chart of Interrupt Handling.....	9-6
11-1	Input and Output Mailbox Interfaces.....	11-6
11-2	Request Message Format.....	11-7
12-1	Syntax Diagrams for Term and Expression.....	12-7
12-2	Syntax Diagram for Item.....	12-7
12-3	Syntax Diagram for Establishing a Breakpoint.....	12-12
12-4	Syntax Diagram for Changing a Breakpoint.....	12-13
12-5	Syntax Diagram for Deleting a Breakpoint.....	12-14
12-6	Syntax Diagram for Examining a Breakpoint.....	12-15
12-7	Syntax Diagram for Viewing the Breakpoint List.....	12-16
12-8	Syntax Diagram for Viewing the Breakpoint Parameters.....	12-17
12-9	Syntax Diagram for Removing a Task From the Breakpoint List.....	12-19
12-10	Syntax Diagram for Establishing the Breakpoint Task.....	12-20
12-11	Syntax Diagram for Inquiring as to the Breakpoint Task.....	12-20
12-12	Syntax Diagram for Viewing the Breakpoint Task's Registers.....	12-22
12-13	Syntax Diagram for Altering the Breakpoint Task's Registers.....	12-23
12-14	Syntax Diagram for Examining or Modifying Memory.....	12-26
12-15	Syntax Diagram for Examining System Objects.....	12-33
12-16	An iRMX 86 Job Report.....	12-34
12-17	An iRMX 86 Task Report.....	12-36
12-18	An iRMX 86 Mailbox Report.....	12-38
12-19	An iRMX 86 Semaphore Report.....	12-39
12-20	An iRMX 86 Segment Report.....	12-40
12-21	Syntax Diagram for Viewing iRMX 86 System Lists.....	12-41
12-22	Syntax Diagram for Exiting the Debugger.....	12-44
12-23	Syntax Diagram for Defining a Numeric Variable.....	12-45
12-24	Syntax Diagram for Changing a Numeric Variable.....	12-45
12-25	Syntax Diagram for Deleting a Numeric Variable.....	12-46
12-26	Syntax Diagram for Viewing Numeric Variables.....	12-46

CHAPTER 1. OVERVIEW

The iRMX 86 Nucleus, Terminal Handler, and Debugger constitute a useful set of iRMX 86 features for development purposes. The Nucleus is required in every application system. The Terminal Handler and Debugger are helpful during development but are frequently omitted thereafter.

NUCLEUS

The Nucleus is the core of every iRMX 86 application system. Among the activities of the Nucleus are the following:

- Supplying scheduling functions.
- Controlling access of tasks to system resources.
- Providing for communication between tasks.
- Enabling the system to respond to external events.

TERMINAL HANDLER

The Terminal Handler provides a real-time, asynchronous interface between your terminal and tasks running under the supervision of the Nucleus. The Terminal Handler provides the following features:

- Line-editing.
- Control characters for suspending and resuming output at the terminal.
- A means of awakening the Debugger.

DEBUGGER

The Debugger is designed specifically for debugging and monitoring systems running under the supervision of the Nucleus. A special debugger is very helpful in debugging such systems, because their real-time and multi-tasking characteristics render inadequate many ordinary debugging techniques. The iRMX 86 Debugger is sensitive to the data structures used by the Nucleus, and it can give "snapshots" of tasks at critical moments, while interfering minimally with the activities of the system being tested. It can also be used to alter the contents of memory.

OVERVIEW

If desired, the Debugger can be included in a debugged application system for troubleshooting in the field. If it is included, the Debugger requires only the support of the Nucleus.

CHAPTER 2. NUCLEUS OVERVIEW

INTRODUCTION

The iRMX 86 Nucleus is one of two major software components of the iRMX 86 Operating System. The other major component, the I/O System, is optional. The Nucleus, however, is required because it is the heart of the system.

The Nucleus provides the building blocks from which the I/O System and application systems are constructed. These building blocks are called objects and are classified into the following five categories called object types:

- Tasks
- Jobs
- Segments
- Mailboxes
- Semaphores

The following simplistic generalizations can be made regarding these types:

- Tasks are the active objects in a system. They do the work of the system.
- Jobs are the environments in which tasks do their work. An environment consists of tasks, the objects that tasks use, and a directory where tasks can catalog objects so as to make them available to other tasks.
- Segments are pieces of memory, the medium that tasks use for communicating.
- Mailboxes are the objects to which tasks go to send or receive segments containing data.
- Semaphores enable tasks to send signals to other tasks.

The Nucleus does extensive record-keeping of objects. It keeps track of each object by means of one or more 16-bit values called tokens. The Nucleus provides a number of operators, called system calls, that tasks use to manipulate objects.

NUCLEUS OVERVIEW

When using a system call, a task supplies parameter values, such as tokens, names, or other values, depending on the requirements of the system call. Some of the functions that tasks can perform with system calls are the following:

- Create objects.
- Delete objects.
- Send messages to other tasks.
- Receive messages from other tasks.
- Obtain information about objects.
- Catalog objects with descriptive names.
- Delete objects from catalogs.

OBJECTS

Each of the five object types has unique characteristics. These characteristics are discussed in detail in the following paragraphs.

TASKS

A task has two goals:

- Its primary goal is to do a specific piece of work.
- Its secondary goal is to obtain exclusive control of the processor so that it can progress toward its primary goal.

One of the main activities of the Nucleus is to arbitrate the competition that results when several tasks each want exclusive control over the processor. The Nucleus does this by maintaining, for each task, an execution state and a priority. The execution state for each task is, at any given time, either running, ready, asleep, suspended, or asleep-suspended. The running state is a special case of the ready state. The priority for each task is an integer value between 0 and 255, inclusive, with 0 being the highest priority.

The arbitration algorithm that the Nucleus uses is that the running task is the ready task with the highest (numerically lowest) priority.

As viewed by the Nucleus, a task is merely a set of values, some of which are the following:

NUCLEUS OVERVIEW

- The task's priority.
- The task's execution state.
- A token for the job that contains the task.

When a task becomes the running task, the following events occur, in order:

- The values of the previously running task are saved by the Nucleus.
- The Nucleus sets the new running task's values.
- The new task begins executing.

The task continues to run until one of the following events occurs:

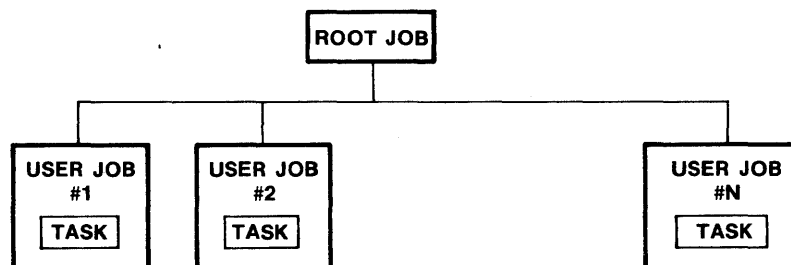
- The task removes itself from the ready state. For example, the task can suspend or delete itself; the task can attempt to receive an object that has not yet been sent, in which case it might elect to wait (in the asleep state).
- The task (task A) is preempted when a higher priority task (task B) becomes ready. An example of how this could happen is that task B might previously have gone into the asleep state for a specific period of time. When the time period has passed, task B becomes ready again. Because it is then the highest priority ready task, task B becomes the running task.

JOB

A job consists of tasks and the resources they need.

The jobs in a system form a family tree, with each job, except the root job, obtaining its resources from its parent. The tasks in the user jobs can create additional objects. If they create additional jobs, this enlarges the job tree.

The job tree, right after the initialization of a system, is as follows:



NUCLEUS OVERVIEW

Associated with each job is an object directory. Objects are known to the Nucleus by their respective tokens, but often, in the code that is executed by tasks, the objects are known by symbolic names. The object directory for a job is a place in memory where a task can catalog an object under a name. Other tasks that know the name can then use the directory to access the object.

SEGMENTS

A fundamental resource that tasks need is memory. Memory is allocated to tasks in the form of segments. A task needing memory requests a segment of whatever size it requires. The Nucleus attempts to create a segment from the memory pool given to the task's job when the job was created.

If there is not enough memory available, the Nucleus will try to get the needed memory from ancestors of the job. In this respect, the tree-structured hierarchy of jobs is instrumental in resource distribution.

MAILBOXES

A mailbox is one of two types of objects that can be used for intertask communication. When task A wants to send an object to task B, task A must send the object to the mailbox, and task B must visit the mailbox, where it has the option of waiting for any desired length of time. Sending an object in this manner can achieve various purposes. It might be a segment that contains data needed by the waiting task. On the other hand, the segment might be blank, and sending it might constitute a signal to the waiting task. Another reason to send an object might be to point out the object to the receiving task.

SEMAPHORES

A semaphore is a custodian of abstract "units". It dispenses units to tasks that request them, and it accepts units from tasks.

An example of typical semaphore use is mutual exclusion. Suppose your application system contains one I/O device which is being used for output by multiple tasks. To ensure that only one of these tasks can use the device at a given time, you can establish a semaphore which has one unit and require that tasks obtain the unit before using the device. A task wanting to use the device would request the unit from the semaphore. When it gets the unit, it can use the device and then return the unit to the semaphore. Because the semaphore has no units while the task is using the device, other tasks are effectively excluded from using the device.

NUCLEUS OVERVIEW

HANDLERS

Two kinds of events can be handled specially. The remainder of this chapter describes the handlers for these events.

EXCEPTION HANDLERS

Tasks occasionally make errors. If an error occurs during an iRMX 86 system call, it causes an exceptional condition. The occurrence of an exceptional condition can, if desired, cause a transfer of control to the exception handler associated with the current task. The exception handler is a procedure that typically deals with the problem by one of the following methods:

- Correcting the cause of the problem and trying again.
- Merely logging the error.
- Deleting the task that caused the error.

In regard to exception handlers, the designer of an iRMX 86-based system has two kinds of decisions that must be made for each task. The first decision concerns the choice of exception handlers. The task can have its own custom exception handler, it can use the exception handler for the job to which it belongs, or it can use the Intel-provided System Exception Handler. Second, there are two categories of exceptional conditions, programmer errors and environmental conditions. Each task can be set up so that control goes to an exception handler in case of

- programmer errors only,
- environmental conditions only,
- in both cases, or
- never.

If control is not directed to an exception handler, the responsibility for handling the exception falls upon the task.

INTERRUPT HANDLERS

To function effectively as a real-time system, an iRMX 86 application system must be responsive to external events. An interrupt handler, one of which is required for each source of external events, is a procedure that is invoked by hardware or software for the purpose of responding to an asynchronous event. The handler takes control immediately and services the interrupt. When the interrupt handler is finished servicing the interrupt, it surrenders the processor, which returns to the interrupted procedure.

CHAPTER 3. JOB MANAGEMENT

A job is an environment in which iRMX 86 objects such as tasks, mailboxes, semaphores, segments, and (offspring) jobs reside. In addition, a job has an object directory and a pool of memory. The job's memory pool provides the raw material from which objects can be created by the tasks in the job. Figure 3-1 illustrates the elements of a job.

Applications consist of one or more jobs. Jobs are independent but they may share resources. Each job has its own tasks and may have its own object directory. Objects may be shared between jobs, although each object is contained in only one job.

The programmer must decide whether tasks belong in the same job. In general, you should place tasks in the same job if:

- they have similar or related purposes
- they share many resources
- they have similar lifespans

JOB TREE AND RESOURCE SHARING

The jobs in a system are arranged in the form of a tree. The root is a job that is provided by the Nucleus. The remaining jobs, including jobs that are created dynamically while the system runs, are descendents of the root job. A job containing tasks that create other jobs is a parent job. A newly created job is a child of the job whose tasks created it.

Associated with each job is a set of limits. The limits of a job are as follows:

- the maximum allowable size of its object directory.
- the maximum and minimum allowable sizes of its memory pool.
- the maximum allowable number of simultaneously existing tasks that it can contain.
- the maximum allowable number of simultaneously existing objects that it can contain.
- the highest allowable priority of any task contained in it.

JOB MANAGEMENT

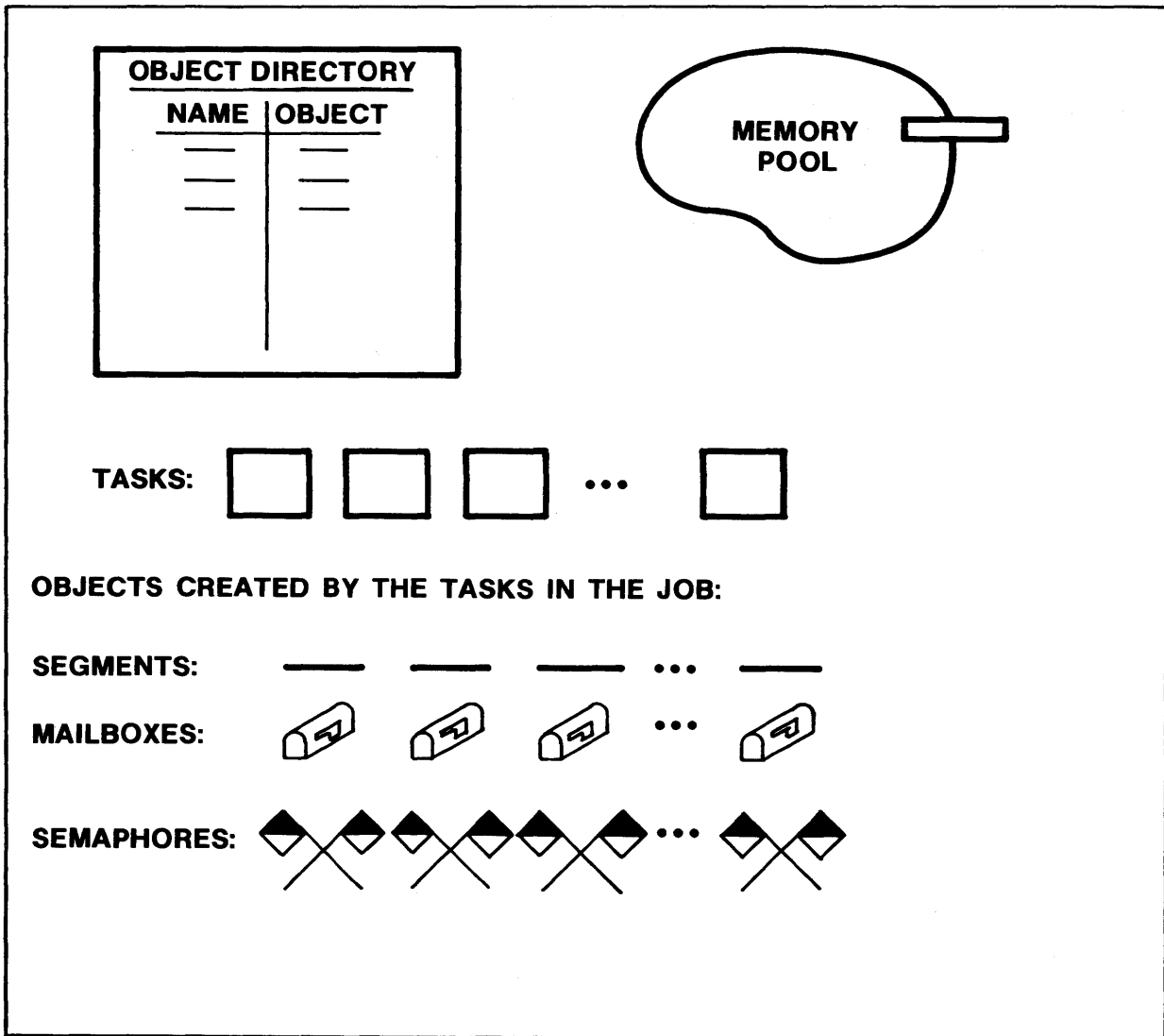


Figure 3-1. A Job

JOB MANAGEMENT

You must specify these limits whenever you create a job and the limits apply collectively to the job and all of its descendent jobs.

When job A creates job B:

- Sufficient memory to meet job B's minimum memory pool requirements is transferred from job A's memory pool to that of job B.
- The memory for job B's object directory is taken from job A's memory pool.
- The numbers of tasks and total objects that job A can contain are reduced by the corresponding values specified for job B.
- The specified maximum priority for tasks in job B cannot exceed the maximum priority for tasks in job A.

If job B is later deleted, its resources are returned to job A.

JOB CREATION

A job is created with one task. The functions of this task include doing some initializing for the new job. Initializing activities can include housekeeping and creating other objects in the new job.

When a task creates a job, it has the option of passing a token for a parameter object to the newly created job. The parameter object can be of any type and it can be used for any purpose. For example, the parameter object might be a segment containing data - arranged in a predefined format - needed by tasks in the new job. Tasks in the new job can obtain a token for the job's parameter object by means of the GET\$TASK\$TOKENS system call, described in Chapter 10.

JOB DELETION

Before a job can be deleted, all of its interrupt tasks (see Chapter 9), extension objects (see the iRMX 86 System Programmer's Reference Manual, number 142721) and descendent jobs must be deleted. By using the OFFSPRING system call, the deleting task can probe down the job tree and find all of the descendents. Then it can delete them, beginning with descendents that have no children and working up the tree. After all of the descendents have been deleted, the task can delete the target job.

JOB MANAGEMENT

SYSTEM CALLS FOR JOBS

The following system calls manipulate jobs:

- CREATE\$JOB --- creates a job with a task and returns a token for the job; resources for the new job are drawn from the resources of the job to which the invoking task belongs.
- DELETE\$JOB --- deletes a childless job that contains no interrupt tasks or extension objects and returns the job's resources to its parent.
- OFFSPRING --- provides a segment containing tokens of the child jobs of the specified job.

CHAPTER 4. TASK MANAGEMENT

Tasks are the active objects in an iRMX 86 system. Each task is part of a job and is restricted to the resources that its job provides. Tasks are written as PL/M-86 procedures, not as main modules.

The iRMX 86 Nucleus maintains a set of attributes for each task. Among these attributes are the priority and execution state of the task. A task's priority is an integer value between 0 and 255, inclusive. The lower the priority number, the higher the priority of the task. A high priority task has favored status as it competes with other tasks for the CPU.

Unless a task is involved in processing interrupts (see Chapter 9), its priority should be between 129 and 255. When a task having a priority in the range 0 to 128 is running, certain external interrupt levels are disabled, depending on the priority.

TASK STATES

A task is always in one of five execution states. The states are asleep, suspended, asleep-suspended, ready, and running.

THE ASLEEP STATE

A task is in the asleep state when it is waiting for a request to be granted. Also, a task can put itself to sleep for a specified amount of time by using the SLEEP system call.

THE SUSPENDED STATE

A task enters the suspended state when it is placed there by another task or when it suspends itself. Associated with each task is a suspension depth, which reflects the number of "suspends" outstanding against it. Each suspend operation must be countered with a resume operation before the task can leave the suspended state.

THE ASLEEP-SUSPENDED STATE

When a sleeping task is suspended, it enters the asleep-suspended state. In effect, it is then in both the asleep and suspended states. While asleep-suspended, the task's sleeping time might expire, putting it in the suspended state.

TASK MANAGEMENT

THE READY AND RUNNING STATES

A task is ready if it is not asleep, suspended, or asleep-suspended. For a task to become the running (executing) task, it must be the highest priority task in the ready state.

TASK STATE TRANSITIONS

The Nucleus does not allocate the processor to tasks in a time-slicing manner. Instead, as an iRMX 86 application system runs, events occur which cause tasks to pass from state to state. The iRMX 86 Operating System is, therefore, event-driven. Figure 4-1 shows the paths of transition between states.

The following list describes, by number, the events that cause the transitions in Figure 4-1. In the list, the migrating task is called "the task":

- (1) The task goes from non-existence to the ready state when it is created.
- (2) The task goes from the ready state to the running state when one of the following occurs:
 - The task has just become ready and has higher priority than does any other ready task.
 - The task is ready, no other ready task has higher priority, no other task of equal priority has been ready for a longer time, and the previously running task has just left the running state by (4), (6), or (10).
- (3) The task goes from the running state to the ready state when the task is preempted by a higher priority task that has just become ready.
- (4) The task goes from the running state to the asleep state when one of the following occurs:
 - the task puts itself to sleep (by the SLEEP system call.)
 - The task makes a request (by the RECEIVE\$MESSAGE, RECEIVE\$UNITS, or LOOKUP\$OBJECT system call) that cannot be granted immediately and expresses, in the request, its willingness to wait.
- (5) The task goes from the asleep state to the ready state or from the asleep-suspended state to the suspended state when one of the following occurs:

TASK MANAGEMENT

- The task's designated waiting period expires without its request being granted.
 - The task's request is granted (because another task called either the SEND\$MESSAGE, SEND\$UNITS, or CATALOG\$OBJECT system call; these calls correspond to those mentioned in (4), above.)
- (6) The task goes from the running state to the suspended state when the task suspends itself (by the SUSPEND\$-TASK system call).

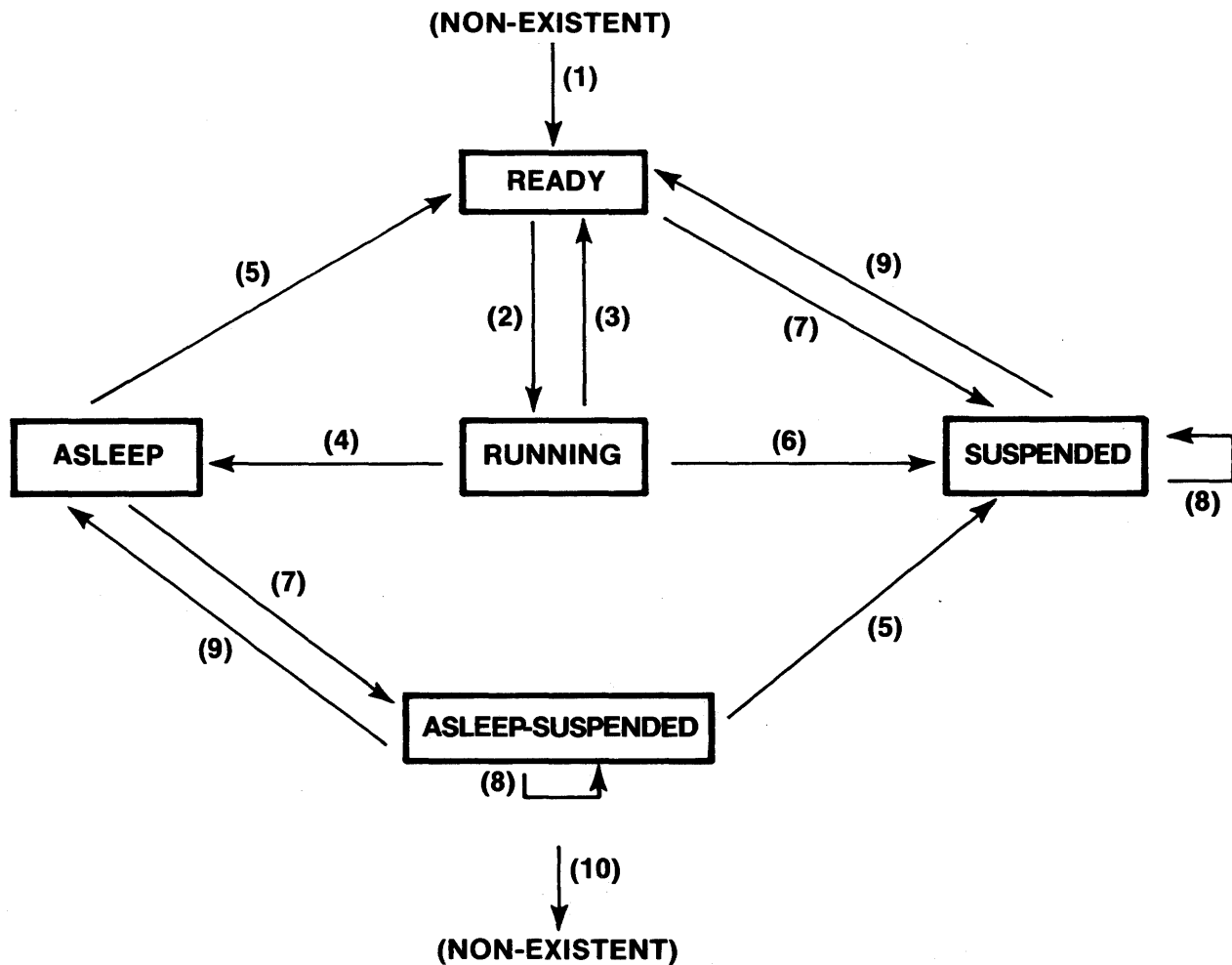


Figure 4-1. Task State Transition Diagram.

TASK MANAGEMENT

- (7) The task goes from the ready state to the suspended state or from the asleep state to the asleep-suspended when the task is suspended by another task (by the SUSPEND\$TASK system call).
- (8) The task remains in the suspended state or the asleep-suspended state when one of the following occurs:
 - (same as (7)) or
 - The task has a suspension depth greater than one and the task is resumed by another task (by the RESUME\$-TASK system call).
- (9) The task goes from the suspended state to the ready state or from the asleep-suspended state to the asleep state when the task has a suspension depth of one and the task is resumed by another task (by the RESUME\$-TASK system call).
- (10) The task goes from any state to non-existence when it is deleted (by the DELETE\$TASK or DELETE\$JOB system call).

ADDITIONAL TASK ATTRIBUTES

In addition to priority, execution state, and suspension depth, the Nucleus maintains current values of the following attributes for each existing task: containing job, its PL/M-86 register context, starting address of its exception handler (see Chapter 8), and exception mode (see Chapter 8).

TASK RESOURCES

When a task is created, the Nucleus takes any resources that it needs at that time (such as a stack segment) from the task's containing job. If the task is subsequently deleted, those resources are returned to the job. The task's code, however, is not a resource in this sense and is not returned.

SYSTEM CALLS FOR TASKS

The following system calls are provided for task manipulation:

- CREATE\$TASK --- creates a task and returns a token for it.
- DELETE\$TASK --- deletes a task from the system.
- SUSPEND\$TASK --- increases a task's suspension depth by one; suspends the task if it is not already suspended.

TASK MANAGEMENT

- RESUME\$TASK --- decreases a task's suspension depth by one; if the depth becomes zero and the task was suspended, it then becomes ready; if the depth becomes zero and the task was asleep-suspended, then it goes into the asleep state.
- SLEEP --- places the calling task in the asleep state for a specified amount of time.
- GET\$TASK\$TOKENS --- returns to the calling task a token for either itself, its job, its job's parameter object, or the root job, depending on which option is specified in the call.
- GET\$PRIORITY --- returns the priority of the specified task.

CHAPTER 5. EXCHANGE MANAGEMENT

The iRMX 86 Nucleus provides exchanges to facilitate intertask communication, synchronization, and mutual exclusion. When a task uses an exchange, it is always acting either as a sender or as a receiver. There are two kinds of exchanges: mailboxes and semaphores. If the exchange is a mailbox, one task will send an object to the mailbox; another task will go to the mailbox to receive the object. If the exchange is a semaphore, either a task is receiving units from the semaphore, or it is sending units to the semaphore.

MAILBOXES

The principal function of mailboxes is to support intertask communication. A sending task uses a mailbox to pass an object to another task. For example, the object might be that of a segment containing data needed by the receiving task.

MAILBOX QUEUES

Each mailbox has two queues, one for tasks that are waiting to receive objects, the other for objects that have been sent by tasks but have not yet been received. The Nucleus sees that waiting tasks receive objects as soon as they are available, so, at any given time, at least one of the mailbox's queues is empty.

MAILBOX MECHANICS

When a task sends a token to a mailbox, using the SEND\$MESSAGE system call, one of two things happens. If no tasks are waiting at the mailbox, the object is placed at the rear of the object queue (which might be empty). Object queues are processed in a first-in-first-out manner, so the object remains in the queue until it makes its way to the front and is given to a task.

If, on the other hand, there are tasks waiting, the receiving task, which has been asleep, goes either from the asleep state to the ready state or from the asleep-suspended state to the suspended state.

EXCHANGE MANAGEMENT

NOTE

If the receiving task has a higher priority than the sending task, then the receiving task preempts the sender and becomes the running task.

When a task attempts to receive an object from a mailbox via the RECEIVE\$MESSAGE system call, and the object queue at the mailbox is not empty, the task receives the object immediately and remains ready. However, if there are no objects at the mailbox there are two possibilities:

- If the task, in its request, elects to wait, it is placed in the mailbox's task queue and is put to sleep. If the designated waiting period elapses before the task gets an object, the task is made ready and receives an E\$TIME exceptional condition (see Chapter 8).
- If the task is not willing to wait, it remains ready and receives an E\$TIME exceptional condition.

A task has the option, when using the SEND\$MESSAGE system call, of specifying that it wants acknowledgment from the receiving task. Thus, any task using the RECEIVE\$MESSAGE system call should check to see if an acknowledgment has been requested. For details, see the description of the RECEIVE\$MESSAGE system call in Chapter 10.

As stated earlier, the object queue for a mailbox is processed in a first-in-first-out manner. However, the task queue of a mailbox can be either first-in-first-out or priority-based, with higher-priority tasks toward the front of the queue. The queuing method to be used is specified for each mailbox at the time of its creation.

SYSTEM CALLS FOR MAILBOXES

The following system calls manipulate mailboxes:

- CREATE\$MAILBOX --- creates a mailbox and returns a token for it.
- DELETE\$MAILBOX --- deletes a mailbox from the system.
- SEND\$MESSAGE --- sends an object to a mailbox.
- RECEIVE\$MESSAGE --- sends the calling task to a mailbox for an object; the task has the option of waiting if no objects are present.

EXCHANGE MANAGEMENT

SEMAPHORES

A semaphore is a custodian of abstract units. A task uses a semaphore either by requesting a specific number of units from it via the RECEIVE\$UNITS system call or by releasing a specific number of units to it via the SEND\$UNITS system call. Although these operations do not support communication of data, they facilitate mutual exclusion, synchronization, and resource allocation.

SEMAPHORE QUEUE

Semaphores have only one queue - a task queue. As is the case with mailboxes, the task queue is either first-in-first-out or priority based. The queueing scheme to be used is specified for each semaphore at the time of its creation.

SEMAPHORE MECHANICS

A semaphore might simultaneously have both tasks in its queue and units in its custody. The allocation scheme used by semaphores is the reason for this. That scheme is best understood by imagining that the semaphore is trying, at all times, to satisfy the request of the task which is at the front of the semaphore's task queue. Only when it can provide as many units as the task requested does it award units, and then it does so immediately.

When a task uses the CREATE\$SEMAPHORE system call, it must supply two values. One value specifies the initial number of units to be in the new semaphore's custody. The other value sets an upper limit on the number of units that the semaphore is allowed to keep at any given time. The lower limit is automatically zero.

When a task requests units from a semaphore via the RECEIVE\$-UNITS system call, the request must be within the specified maximum for that semaphore; otherwise, the request is invalid and causes an E\$LIMIT exceptional condition. If a task's request for units is valid and both

- the size of the request is within the semaphore's current supply of units and
- the task is - or would be if queued - at the front of the semaphore's task queue,

then the request is granted immediately and the task remains ready. Otherwise, one of the following applies:

EXCHANGE MANAGEMENT

- The task, in its request, elects to wait. It is placed in the semaphore's task queue and is put to sleep. If the designated waiting period elapses before the task gets its requested units, the task is made ready and receives an E\$TIME exceptional condition.
- The task is not willing to wait. It remains ready and receives an E\$TIME exceptional condition.

Suppose, for example, that two tasks, A and B, are waiting at a semaphore, with A at the front of the queue. The semaphore has no units, A wants 3 units, and B wants 1 unit. The following three separate cases illustrate the mechanics of the semaphore:

- If the semaphore is sent 2 units, both A and B remain asleep in the semaphore's queue. Note that B's modest request is not satisfied because A is ahead of B in the queue.
- If, instead, the semaphore is sent 3 units, A receives the units and awakens, while B remains asleep in the queue.
- If, instead, the semaphore is sent 4 units, A and B both receive their requested units and are awakened.

When a task sends units to a semaphore, the task remains ready. Sending units to a semaphore causes an E\$LIMIT exceptional condition if it pushes the semaphore's supply above the designated maximum. The number of units in the custody of the semaphore remains unchanged.

NOTE

It is possible that a task sending units to a semaphore can be preempted by a higher priority task becoming ready as a result of getting its requested units.

SYSTEM CALLS FOR SEMAPHORES

The following system calls manipulate semaphores:

- CREATE\$SEMAPHORE --- creates a semaphore and returns a token for it.
- DELETE\$SEMAPHORE --- deletes a semaphore from the system.
- SEND\$UNITS --- adds a specific number of units to the supply of a semaphore.
- RECEIVE\$UNITS --- asks for a specific number of units from a semaphore.

CHAPTER 6. MEMORY MANAGEMENT

Occasionally a task needs additional memory, that is, memory not yet allocated to its job. By using Nucleus system calls for allocating and deallocating memory, tasks can usually satisfy their memory needs.

SEGMENTS

Allocated memory is treated as a collection of segments. A segment is a contiguous sequence of 16 byte paragraphs, with its starting (base) address evenly divisible by 16. The Nucleus maintains, as attributes, the base address and the length in bytes of each segment.

When a task needs a segment, it can request one of the desired length via the CREATE\$SEGMENT system call. If enough memory is available, the Nucleus returns a token for the segment.

NOTE

The token of a segment can be used as the base portion of a pointer to the segment. Thus, the token can be used as a base address (as when writing a message in the segment) or as an object reference (as when sending the segment-with-message to a mailbox).

MEMORY POOLS

A memory pool is the amount of memory available to a job and its descendents. Each job has a memory pool. When a job is created, the memory for its pool is borrowed from the pool of its parent job. Thus, there is effectively a tree-structured hierarchy of memory pools, identical in structure to the hierarchy of jobs. Memory that a job borrows from its parent remains in the pool of the parent as well as being in the pool of the child. Such memory, however, is available for use by tasks in the child job, but not by tasks in the parent job. Figure 6-1 illustrates the relationship between the job and memory hierarchies. In the figure, the pool sizes shown are actually the maximum sizes of those pools.

MEMORY MANAGEMENT

CONTROLLING POOL SIZE

Two parameters, pool minimum and pool maximum, of the CREATE\$-JOB system call, dictate the range of sizes (in paragraphs) of a new job's memory pool. Initially, the pool size is equal to pool minimum. Memory allocated to tasks in the job is still considered to be in the job's pool. A task needing to know

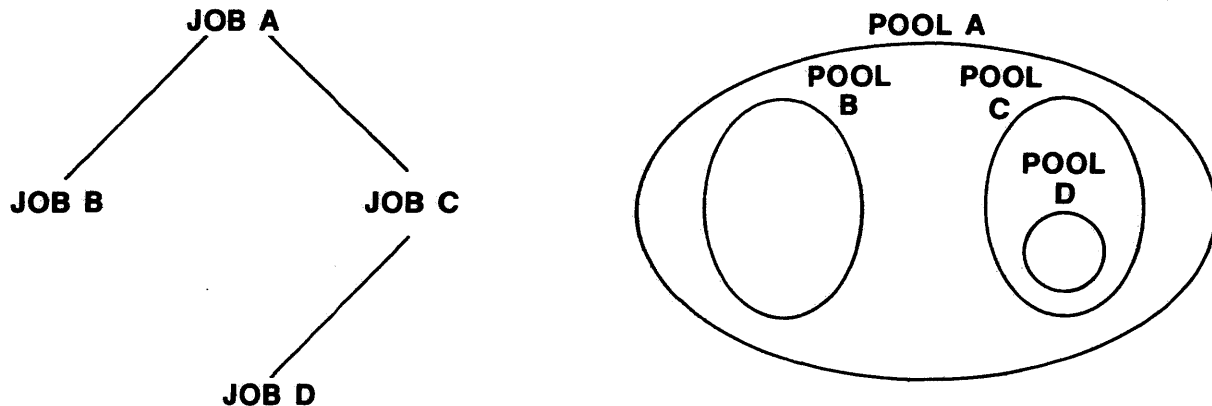


Figure 6-1. Comparison of Job and Memory Hierarchies

about its job's pool may use the GET\$POOL\$ATTRIB system call to obtain pool minimum, pool maximum, initial pool size, number of bytes currently available, and number of bytes currently allocated.

A task may alter the pool minimum attribute for its job by means of the SET\$POOL\$MINIMUM system call; pool minimum must lie in the range from 0 to pool maximum. If a subsequent call to SET\$POOL\$MINIMUM increases the pool's minimum size, and the current pool size is less than the new minimum, no memory is borrowed immediately from the parent job. Rather, memory is automatically borrowed as it is requested by tasks in the job, until the new minimum is reached. At that time, the new value of the pool minimum attribute becomes a lower bound for the job's pool size.

MOVEMENT OF MEMORY BETWEEN JOBS

When a task tries to create a segment, and the unallocated part of its job's pool is not sufficient to satisfy the request, the Nucleus tries to borrow more memory from the job's parent (and

MEMORY MANAGEMENT

then, if necessary, from its parent's parent, and so on). Such borrowing increases the pool size of the borrowing job and is thus restricted by the pool maximum attribute of the borrowing job.

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

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

MEMORY ALLOCATION

The memory pool of a job consists of two classes of memory: allocated and unallocated. Memory in a job is allocated if it has been requested by tasks in the job or if it is on loan to a child job. Otherwise, it is unallocated.

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

When a task no longer needs a segment, it can return the segment to the unallocated part of the job's pool by using the DELETE\$SEGMENT system call. Figure 6-2 shows how memory "moves."

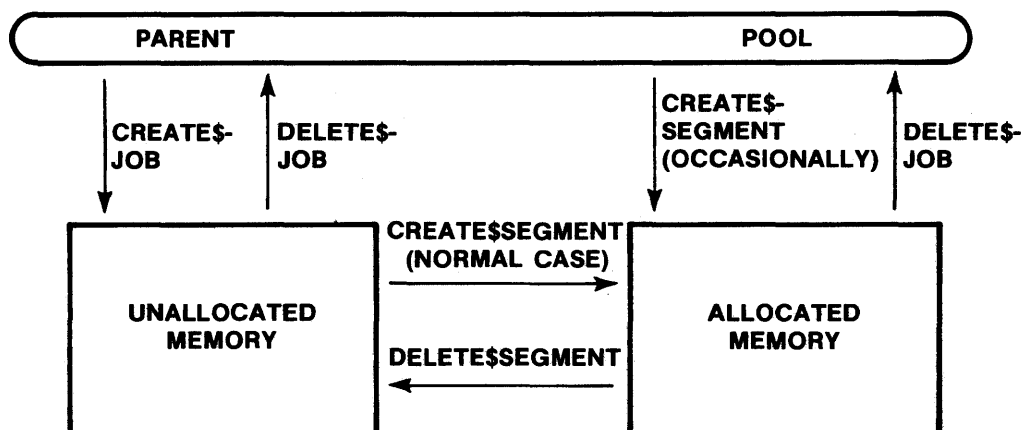


Figure 6-2. Memory Movement Diagram

MEMORY MANAGEMENT

SYSTEM CALLS FOR SEGMENTS

The following system calls manipulate segments:

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

CHAPTER 7. OBJECT MANAGEMENT

A few iRMX 86 Nucleus system calls apply to all object types. One of these, the GET\$TYPE system call, enables a task to present a token to the Nucleus and get an object's type code in return. (Type codes are listed in Appendix B.) This is useful, for example, when a task is expecting to receive objects of several different types. With the object's type code, the task can use the appropriate system calls for the object.

Other type-independent system calls have to do with object directories. Each job has its own object directory. An entry in an object directory consists of an object with an ASCII name. Such a feature is often needed because some tasks might only know some objects by their associated names.

By using the LOOKUP\$OBJECT system call, a task can present the name of an object to the Nucleus. The Nucleus consults the object directory corresponding to the specified job and, if the object has been cataloged there, returns the token.

NOTE

In object directory lookups, upper and lower case alphabetic characters are treated as being different.

If the object has not yet been cataloged, and the task is not willing to wait, the task remains ready and receives an E\$TIME exceptional condition. However, if the task is willing to wait, it is put to sleep; there are two possibilities:

- If the designated waiting period elapses before the task gets its requested token, the task is made ready and receives an E\$TIME exceptional condition (see Chapter 8).
- If the task gets its requested token within the designated waiting period, it is made ready with no exceptional condition. This case is possible because another task can, while the requesting task is waiting, catalog the appropriate entry in the specified object directory.

OBJECT MANAGEMENT

The tasks in a job must maintain the job's object directory. When a task wants to share an object with the other tasks in a job (not necessarily its own job), it can use the CATALOG\$-OBJECT system call to put the object in that job's object directory. Typically, this is done by the creator of the object. Likewise, entries can be removed from a directory by the UNCATALOG\$OBJECT system call.

What is required, when using an object directory, is the token of the job whose directory is to be used. The root job's object directory, called the root object directory, is special in that any task can use it. Any task can call the GET\$TASK\$-TOKENS system call to obtain the token of the root job.

SYSTEM CALLS FOR ANY OBJECTS

The following system calls manipulate objects:

- CATALOG\$OBJECT --- places an object in an object directory.
- UNCATALOG\$OBJECT --- removes an object from an object directory.
- LOOKUP\$OBJECT --- accepts a cataloged name of an object and returns a token for it.
- GET\$TYPE --- accepts a token for an object and returns its type code.

CHAPTER 8. EXCEPTIONAL CONDITION MANAGEMENT

When a task invokes an iRMX 86 system call, the results are sometimes not what the task is trying to achieve. For example, maybe the task requests memory that is not available, or it might use an invalid token as a parameter. In such cases, the system must inform the task that an error occurred. Whenever a task makes a system call, the means of communicating the success or failure of the call is the condition code.

TYPES OF EXCEPTIONAL CONDITIONS

Table 8-1 is a list of Nucleus conditions and their codes. The conditions that represent failure are called exceptional and are classified as programmer errors or environmental conditions. An exceptional condition that is preventable by the calling task is a programmer error. In contrast, exceptional conditions due to environmental circumstances of which the task could have no awareness are considered environmental conditions.

Table 8-1 lists the possible conditions, with their associated numeric codes and mnemonics. Values not used as numeric codes are reserved.

EXCEPTION HANDLERS

The iRMX 86 Nucleus supports exception handlers. Their purpose is to deal with the errors that tasks make in making system calls. How an exception handler deals with an exceptional condition is a matter of programmer discretion. In general, a handler performs one of the following actions:

- Logs the error.
- Deletes the task that erred.
- Simply ignores the error. If this option is taken, the system continues as if no error had occurred. Continuing under such circumstances is generally unwise, however.

An exception handler is written as a procedure with four parameters passed in the following order:

- the condition code (WORD).
- a code (BYTE) indicating which parameter, if any, was faulty in the call (1 for first, 2 for second, etc., 0 if none).

EXCEPTIONAL CONDITION MANAGEMENT

- a reserved (WORD) parameter.
- a second reserved (WORD) parameter.

ASSIGNING AN EXCEPTION HANDLER

A task may use the SET\$EXCEPTION\$HANDLER system call to declare its own exception handler. Otherwise, the task inherits the exception handler of its job. A job can receive its own exception handler at the time of its creation. If it doesn't, the job inherits the system exception handler. Thus, the Nucleus can always find an exception handler for the running task.

A system exception handler is provided as part of the iRMX 86 Operating System and deletes any task on whose behalf it is invoked.

Users wanting to write their own exception handlers should compile them under the PL/M-86 LARGE control.

Any task can have the Debugger as its exception handler; see the description in Chapter 10 of the SET\$EXCEPTION\$HANDLER system call for instructions on how to dynamically make such an assignment. Alternatively, the Debugger or any other routine can be made the system exception handler statically; see the iRMX 86 Configuration Guide for ISIS-II Users, number 9803126, for information on how to do this.

INVOKING AN EXCEPTION HANDLER

When a task causes an exceptional condition, it need not have control passed to its exception handler. The factor that determines whether control passes to the exception handler is the task's exception mode. This attribute has four possible values, each of which specifies the circumstances under which the exception handler is to get control in the event of an exceptional condition. These circumstances are:

- Programmer errors only.
- Environmental conditions only.
- All exceptional conditions.
- No exceptional conditions.

When the Nucleus detects that a task has caused an exceptional condition in making a system call, it compares the type of the condition with the calling task's exception mode. If a transfer of control is indicated, the Nucleus passes control to the exception handler on behalf of the task. The exception

EXCEPTIONAL CONDITION MANAGEMENT

Table 8-1. Conditions and Their Codes

CATEGORY/ MNEMONIC	MEANING	NUMERIC CODE	
		HEX	DECIMAL
Normal			
E\$OK	The most recent system call was successful.	0H	0
Exceptional			
Environmental Conditions			
E\$TIME	A time limit (possibly a limit of zero time) expired without a task's request being satisfied.	1H	1
E\$MEM	There is not sufficient memory available to satisfy a task's request.	2H	2
E\$LIMIT	A task attempted an operation which, if it had been successful, would have violated a Nucleus-enforced limit.	4H	4
E\$CONTEXT	A system call was issued out of context.	5H	5
E\$EXIST	A token parameter has a value which is not the token of an existing object.	6H	6
E\$STATE	A task attempted an operation which would have caused an impossible transition of a task's state.	7H	7
E\$NOT\$CON- FIGURED	The system call being attempted is not part of the present software configuration.	8H	8
Programmer Errors			
E\$ZERO\$- DIVIDE	A task attempted to divide by zero.	8000H	32768
E\$OVERFLOW	An overflow interrupt occurred.	8001H	32769
E\$TYPE	A token parameter referred to an existing object that is not of the required type.	8002H	32770
E\$PARAM	A parameter which is neither a token nor an offset has an illegal value.	8004H	32772
E\$BAD\$CALL	A task attempted a software interrupt to level 184.	8005H	32773

EXCEPTIONAL CONDITION MANAGEMENT

handler then deals with the problem, after which control returns to the task, unless the exception handler deleted the task. While the exception handler is executing, the errant task is still regarded by the Nucleus to be the running task.

When a task is created, its exception mode is set to its job's default exception mode. The task can change its exception handler and exception mode attributes by using the SET\$-EXCEPTION\$HANDLER system call.

HANDLING EXCEPTIONS IN-LINE

If a task's exception mode attribute does not direct the Nucleus to transfer control to the task's exception handler, the responsibility for dealing with an error falls upon the task.

Each system call has as its last parameter a pointer to a WORD. After a system call, the Nucleus returns the resulting condition code to this WORD. By checking this WORD after each system call, a task can ascertain whether the call is successful. (See Table 8-1 for condition codes.) If the call is not successful, the task can learn which exceptional condition it caused. This information can sometimes enable the task to recover. In other cases more information is needed.

NOTE

If an exceptional condition is caused by an invalid parameter, an exception handler, which is passed the parameter number of the first invalid parameter, should handle the condition.

SYSTEM CALLS FOR EXCEPTION HANDLERS

The following system calls manipulate exception handlers:

- SET\$EXCEPTION\$HANDLER --- sets the exception handler and exception mode attributes of the calling task.
- GET\$EXCEPTION\$HANDLER --- returns to the calling task the current values of its exception handler and exception mode attributes.

CHAPTER 9. INTERRUPT MANAGEMENT

Interrupts and interrupt processing are central to real-time computing. External events occur asynchronously with respect to the internal workings of a iRMX 86 application system. An interrupt, signalling the occurrence of an external event, triggers an implicit "call" to a specific location in a section of memory known as the interrupt vector table. From there, control is redirected to a PL/M-86 interrupt procedure called an interrupt handler. At this point, one of two things happens. If handling the interrupt takes little time and requires no system calls, other than certain interrupt-related system calls, the interrupt handler processes the interrupt. Otherwise, the interrupt handler invokes an interrupt task which deals with the interrupt. After the interrupt has been serviced, control returns to the ready application task with highest priority.

INTERRUPT MECHANISMS

There are three major concepts in interrupt processing: the interrupt vector table, interrupt levels, and disabling interrupt levels.

THE INTERRUPT VECTOR TABLE

The interrupt vector table is composed of 256 vectors. The vectors are numbered 0 to 255. A number of the interrupt vectors are reserved and therefore are not available to be defined by user tasks. The vectors are allocated as follows:

- 0- 55: reserved
- 56- 63: available for external interrupts
- 64-223: reserved
- 224-255: described in the iRMX 86 System Programmer's Reference Manual.

INTERRUPT LEVELS

External interrupts are funneled through hardware which can manage interrupts from up to eight external sources. The eight sources are associated with eight interrupt levels.

INTERRUPT MANAGEMENT

The interrupt levels, numbered 0 to 7, correspond to interrupt vectors 56 to 63, respectively. Interrupt levels with low numbers have high priority. As a rule, all levels except level 2 are available for user devices. Level 2 is reserved for the system clock.

DISABLING INTERRUPTS

Occasionally you want to prevent interrupt signals from causing an immediate interrupt. For example, it is desirable to prevent low priority interrupts from interfering with the servicing of a high priority interrupt. In the iRMX 86 Operating System, each interrupt level can be disabled. In some circumstances, described later, the Nucleus disables levels. Tasks can also disable and enable levels by means of the DISABLE and ENABLE system calls. Level 2, which is reserved for the system clock, should not be disabled.

If an interrupt signal arrives at a level that is enabled, the interrupt is recognized by the processor and control goes immediately to the interrupt handler for that level. Otherwise, the level is disabled and the interrupt signal is blocked until the level is enabled, at which time the signal is recognized by the CPU. However, if the signal is no longer emanating from its source, it is not recognized and the interrupt is not handled.

There are two ways in which an interrupt level can be disabled. A task can mask a level by using the DISABLE system call; later the level can be unmasked by the ENABLE system call. The second disabling agent is the Nucleus itself. When the running task is of high priority, the Nucleus disables certain interrupt levels. The relationship between task priorities and disabled levels is given in Table 9-1.

Table 9-1. Interrupt Levels Disabled For Running Task

Task Priority	Disabled Levels
0-16	0-7
17-32	1-7
33-48	2-7
49-64	3-7
65-80	4-7
81-96	5-7
97-112	6-7
113-128	7
129-255	None

INTERRUPT MANAGEMENT

INTERRUPT HANDLERS AND INTERRUPT TASKS

Whether an interrupt level is to be serviced by an interrupt handler alone or by having an interrupt handler invoke an interrupt task depends on two conditions. First, interrupt handlers cannot make most system calls. Only ENTER\$INTERRUPT, EXIT\$INTERRUPT, GET\$LEVEL, and SIGNAL\$INTERRUPT can be called from an interrupt handler. If other system calls are required, they must be made by an interrupt task. Second, an interrupt handler should call an interrupt task unless it can service interrupts quickly because an interrupt signal disables all interrupts, and interrupts remain disabled until the interrupt handler is finished processing. Invoking an interrupt task, on the other hand, allows higher priority interrupts to be accepted.

SETTING UP AN INTERRUPT HANDLER

Interrupt handlers are generally written as PL/M-86 interrupt procedures, but can be written in assembly language. If an interrupt handler is written in assembly language, it must save and restore all register values, as noted later.

An interrupt handler and, optionally, an interrupt task, are bound to an interrupt level by means of a SET\$INTERRUPT system call. SET\$INTERRUPT places the starting address of an interrupt handler in the interrupt vector table. If the interrupt\$task\$flag parameter in SET\$INTERRUPT is set to 1, the calling task becomes the interrupt task for the specified level. Otherwise, interrupt\$task\$flag is 0 and there is no interrupt task for the level.

Any desired value can be specified as the data segment base address for an interrupt handler by means of the interrupt\$handler\$ds parameter in SET\$INTERRUPT. The interrupt handler can later cause this value to be loaded into the DS register by calling ENTER\$INTERRUPT.

When an iRMX 86 application system starts up, all interrupt levels are disabled. When SET\$INTERRUPT binds an interrupt handler but not an interrupt task to a level, the level is enabled. If, instead, there is an interrupt task, the level is not enabled until that task makes a WAIT\$INTERRUPT system call (described later.) An interrupt task should not enable its own level before making its first call to WAIT\$INTERRUPT.

A RESET\$INTERRUPT system call cancels the bond between an interrupt level and its interrupt handler. The call also disables the specified level. If there is an interrupt task for the level, RESET\$INTERRUPT deletes it. DELETE\$TASK and DELETE\$JOB do not delete interrupt tasks.

INTERRUPT MANAGEMENT

USING AN INTERRUPT HANDLER

If an interrupt handler is to service interrupts for a given level without invoking an interrupt task, the handler must assume one of two forms, depending on whether it needs to have the Nucleus set up its data segment base address.

If the interrupt handler can use the data segment base address of the interrupted task or if it contains its data segment base address in its code, then it should perform the following functions in the following order:

- If in assembly language, save all register contents
- Service the interrupt
- Call EXIT\$INTERRUPT
- If in assembly language, restore all register contents

The call to EXIT\$INTERRUPT sends an end-of-interrupt signal to the hardware.

If the interrupt handler wants the Nucleus to load a data segment base address (as specified in an earlier call to SET\$INTERRUPT) into the DS register, then it should perform the following functions in the following order:

- If in assembly language, save all register contents
- Optionally, do some interrupt servicing
- Call ENTER\$INTERRUPT
- Complete interrupt servicing
- Call EXIT\$INTERRUPT
- If in assembly language, restore all register contents

The call to ENTER\$INTERRUPT tells the Nucleus to load the interrupt handler's data segment base address into the DS register. Because PL/M-86 makes use of the data segment, as specified by the contents of the DS register, loading a new value into this register serves to protect the data segment of the interrupted task.

USING AN INTERRUPT TASK

If there is both an interrupt handler and an interrupt task associated with a level, the interrupt handler invokes the interrupt task by making a SIGNAL\$INTERRUPT system call. If a level has only an interrupt handler, however, the handler may not call SIGNAL\$INTERRUPT.

If an interrupt handler calls an interrupt task, the handler must perform the following functions in the following order:

INTERRUPT MANAGEMENT

If in assembly language, save the register contents.
Optionally, call ENTER\$INTERRUPT.
Optionally, begin servicing the interrupt without system calls.
Call SIGNAL\$INTERRUPT.
If in assembly language, restore the register contents.

The call to SIGNAL\$INTERRUPT starts up the interrupt task and enables interrupts.

If used, the call to ENTER\$INTERRUPT sets up a new DS value for the interrupt handler. If you want the interrupt handler to have the same DS value as that used by the interrupt task, so the handler can pass data to the task, follow the advice given in the description of the interrupt\$handler\$ds parameter of SET\$INTERRUPT in Chapter 10.

An interrupt handler executes in the environment (except possibly for the DS register value) of the interrupted task. The interrupt task, however, like any other task, has its own environment.

An interrupt task must perform the following functions in the following order, although the first two functions may be interchanged:

```
Call SET$INTERRUPT.  
Do initialization.  
Do forever;  
    Call WAIT$INTERRUPT.  
    Service the interrupt (system calls allowed).  
End;
```

An interrupt task, once initialized, is always in one of two modes. Either it is servicing an interrupt or it is waiting for notification of an interrupt.

When a task becomes an interrupt task by calling SET\$INTERRUPT, the Nucleus assigns a priority to it, according to the level that the task is to service. Table 9-2 shows the relationship between levels and interrupt task priorities.

NOTES

The priority that the Nucleus assigns to an interrupt task might exceed the maximum priority attribute of the job that contains that task. If this occurs, you get an exceptional condition. To overcome this problem, recreate the job with a higher maximum priority attribute.

INTERRUPT MANAGEMENT

Because the automatic filling of the interrupt vector is overridden by SET\$INTERRUPT, the NOINTVECTOR control should be used when compiling the interrupt handler.

Figure 9-1 illustrates the two interrupt servicing patterns and their relationships.

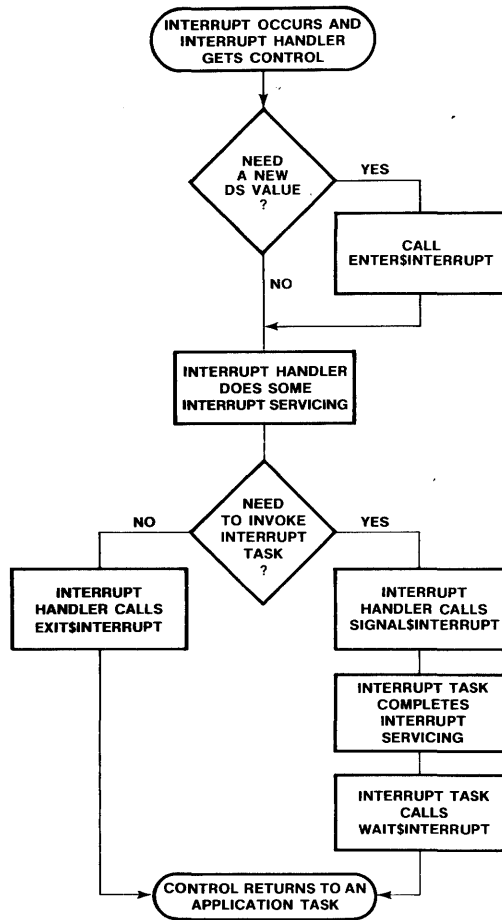


Figure 9-1. Flow Chart of Interrupt Handling

Note that an interrupt handler might call an interrupt task sometimes yet not call it at other times. An example is an interrupt handler that puts characters into a buffer for a line printer. Whenever a character is received, the interrupt handler is invoked and puts the character in the line buffer. If the character is an end-of-line character, or if the character count maintained by the interrupt handler indicates that the buffer is full, the interrupt handler calls its interrupt task, which outputs the contents of the buffer. Otherwise, the interrupt handler calls EXIT\$INTERRUPT and then returns control to application tasks.

INTERRUPT MANAGEMENT

Table 9-2. The Relationship Between External Levels and Internal Task Priorities

<u>LEVEL</u>	<u>INTERRUPT TASK PRIORITY</u>
0	18
1	34
2	50
3	66
4	82
5	98
6	114
7	130

HANDLING LEVEL 7 INTERRUPTS

Occasionally, spurious signals can trigger a level 7 interrupt. An interrupt handler for level 7 should begin by sampling port CO. If the BYTE value obtained there does not have a 1 in the high-order bit, then the interrupt is a false alarm and should not be handled. In PL/M-86, the following lines perform this check when placed at the beginning of the interrupt handler:

```
OUTPUT (COH) = 0BH;
IF ((INPUT (COH)) AND 80H) = 0
    THEN RETURN;
```

EXAMPLES OF INTERRUPT SERVICING

To help you understand the major points already described, tables 9-3 and 9-4 are provided. Each table outlines the turning points in a scenario where an interrupt handler is assigned to level 4, an interrupt arrives at that level and is serviced, and finally the assignment of an interrupt handler is cancelled. Table 9-3 shows a case where the interrupt handler deals with the interrupt. Table 9-4 treats the case where the interrupt handler calls an interrupt task.

In the right-hand column of each of tables 9-3 and 9-4, the phrase "interrupt levels necessarily disabled" alludes to the fact that the events of the example cause certain levels to be enabled or disabled. Other events, outside the scope of the example, might cause other levels to be disabled as well.

INTERRUPT MANAGEMENT

Table 9-3. An Example Of Interrupt Handling Without An Interrupt Task

EVENTS IN SEQUENCE	EXPLANATION	INTERRUPT LEVELS NECESSARILY DISABLED
-	No interrupt handler assigned to level 4.	4
RQ\$SET\$INTERRUPT (LEVEL\$4,0,...);	A task assigns an interrupt handler to level 4.	NONE
Level 4 device interrupts	An interrupt arrives at level 4.	0-7
.	The interrupt is serviced by the interrupt handler.	0-7
RQ\$EXIT\$INTERRUPT (LEVEL\$4,...);	Interrupt hardware reset by the interrupt handler.	0-7
Interrupt handler returns	Interrupts are re-enabled.	NONE
RQ\$RESET\$INTERRUPT- (LEVEL\$4,...);	A task cancels the assignment of an interrupt handler to level 4.	4

INTERRUPT MANAGEMENT

Table 9-4. An Example Of Interrupt Handling
With An Interrupt Task

EVENTS IN SEQUENCE	EXPLANATION	INTERRUPT LEVELS NECESSARILY DISABLED
-	No interrupt handler assigned to level 4.	4
RQ\$SET\$INTERRUPT (LEVEL\$4, 1, ...);	A task assigns an interrupt handler to level 4 and it assigns itself to be the interrupt task for that level.	4
RQ\$WAIT\$INTERRUPT (LEVEL\$4,...);	The interrupt task begins to wait for an interrupt.	NONE
Level 4 device interrupts	An interrupt arrives at level 4. The interrupt handler gets control and optionally, does some servicing.	0-7
RQ\$SIGNAL\$INTERRUPT (LEVEL\$4,...);	The interrupt handler invokes the interrupt task.	4-7
.	The interrupt is serviced by the interrupt task.	4-7
.		
.		
RQ\$WAIT\$INTERRUPT (LEVEL\$4,...);	The interrupt task finishes and begins to wait for another level 4 interrupt. Control passes back to the interrupt handler and then back to an application task.	NONE

INTERRUPT MANAGEMENT

SYSTEM CALLS FOR INTERRUPTS

The following system calls manipulate interrupts:

- SET\$INTERRUPT --- assigns an interrupt handler and, if desired, an interrupt task to an interrupt level.
- RESET\$INTERRUPT --- cancels the assignment made to a level by SET\$INTERRUPT and, if applicable, deletes the interrupt task for that level.
- EXIT\$INTERRUPT --- used by interrupt handlers to send an end-of-interrupt signal to hardware.
- SIGNAL\$INTERRUPT --- used by interrupt handlers to invoke interrupt tasks.
- WAIT\$INTERRUPT --- puts the calling interrupt task to sleep until it is called into service by an interrupt handler.
- ENABLE --- enables an external interrupt level.
- DISABLE --- disables an external interrupt level.
- GET\$LEVEL --- returns the interrupt level of highest priority for which an interrupt handler has started but has not yet finished processing.
- ENTER\$INTERRUPT --- sets up a previously designated data segment base address for the calling interrupt handler.

CHAPTER 10. NUCLEUS SYSTEM CALLS

This chapter contains the calling sequences and other information about the system calls to the Nucleus. The system calls are listed in alphabetical order. Names of the calls are written in white on a dark background in the upper outside corner of each page. The calling sequence for each call is that for the PL/M-86 interface. The information for each system call is organized into the following categories, in the following order:

- A brief sketch of the effects of the call.
- The format of the call.
- Definitions of the input parameters, if any.
- Definitions of the output parameters, if any.
- A complete description of the effects of the call.
- The condition codes that can result from using the call, with a description of the possible causes of each condition.

Throughout the chapter, iRMX 86 data types, such as BYTE, STRING are used. They are always capitalized and their definitions are found in Appendix A.

Between this introduction and the details of the system calls is a command dictionary, in which the calls are grouped according to type. This dictionary, which includes short descriptions and page numbers of the complete descriptions in this chapter, is provided as an alternate way of indexing the system calls.

NUCLEUS SYSTEM CALLS

COMMAND DICTIONARY

CALLS FOR JOBS

- CREATE\$JOB -- Creates a job with a task and returns
a token for the job 10-7
- DELETE\$JOB -- Deletes a childless job that contains
no interrupt tasks or extension objects..... 10-20
- OFFSPRING -- Provides a segment containing tokens of
the child jobs of the specified job 10-42

CALLS FOR TASKS

- CREATE\$TASK -- Creates a task and returns a token for it 10-17
- DELETE\$TASK -- Deletes a task 10-25
- SUSPEND\$TASK -- Increases a task's suspension depth by
one; suspends the task if it is not already suspended . 10-61
- RESUME\$TASK -- Decreases a task's suspension depth by
one; resumes (unsuspends) the task if the suspension
depth becomes zero 10-48
- SLEEP -- Places the calling task in the asleep state
for a specified amount of time 10-60
- GET\$TASK\$TOKENS -- Returns to the caller a token for
either itself, its job, its job's parameter object,
or the root job 10-38
- GET\$PRIORITY -- Returns the priority of a task 10-36

CALLS FOR MAILBOXES

- CREATE\$MAILBOX -- Creates a mailbox and returns a token
for it 10-13
- DELETE\$MAILBOX -- Deletes a mailbox 10-22
- SEND\$MESSAGE -- Sends an object to a mailbox 10-49
- RECEIVE\$MESSAGE -- Sends the calling task to a mailbox
for an object; the task has the option of waiting
if no objects are present 10-43

NUCLEUS SYSTEM CALLS

COMMAND DICTIONARY (continued)

CALLS FOR SEMAPHORES

- CREATE\$SEMAPHORE -- Creates a semaphore and returns
a token for it 10-15
- DELETE\$SEMAPHORE -- Deletes a semaphore 10-24
- SEND\$UNITS -- Adds a specific number of units to the
supply of a semaphore 10-51
- RECEIVE\$UNITS -- Asks for a specific number of units
from a semaphore 10-45

CALLS FOR SEGMENTS AND MEMORY POOLS

- CREATE\$SEGMENT -- Creates a segment and returns a token
for it 10-14
- DELETE\$SEGMENT -- Returns a segment to the memory pool
from which it was allocated 10-23
- GET\$SIZE -- returns the size, in bytes, of a segment 10-37
- SET\$POOL\$MINIMUM -- Changes the pool minimum attribute
of the memory pool of the caller's job 10-58
- GET\$POOL\$ATTRIBUTES -- Returns the following memory pool
attributes of the caller's job: pool minimum, pool
maximum, initial size, number of allocated bytes,
number of available bytes 10-34

CALLS FOR ALL OBJECTS

- CATALOG\$OBJECT -- Places an object in an object
directory 10-5
- UNCATALOG\$OBJECT -- Removes an object from an object
directory 10-62
- LOOKUP\$OBJECT -- Accepts a cataloged name of an object
and returns a token for it 10-40
- GET\$TYPE -- Accepts a token for an object and returns
its type code 10-39

NUCLEUS SYSTEM CALLS

COMMAND DICTIONARY (continued)

CALLS FOR EXCEPTION HANDLERS

- SET\$EXCEPTION\$HANDLER -- Sets the exception handler and exception mode attributes of the caller 10-52
- GET\$EXCEPTION\$HANDLER -- Returns the current values of the caller's exception handler and exception mode attributes 10-31

CALLS FOR INTERRUPT HANDLERS, TASKS, AND LEVELS

- SET\$INTERRUPT -- Assigns an interrupt handler and, if desired, an interrupt task to an interrupt level 10-54
- RESET\$INTERRUPT -- Cancels the assignment of an interrupt handler to a level and, if applicable, deletes the interrupt task for that level 10-47
- ENTER\$INTERRUPT -- Sets up a previously designated data segment base address for the calling interrupt handler..... 10-28
- EXIT\$INTERRUPT -- Used by interrupt handlers to send an end-of-interrupt signal to hardware 10-30
- SIGNAL\$INTERRUPT -- Used by interrupt handlers to invoke interrupt tasks 10-59
- WAIT\$INTERRUPT -- Puts the calling interrupt task to sleep until it is called into service by an interrupt handler10-63
- ENABLE -- Enables an external interrupt level 10-27
- DISABLE -- Disables an internal interrupt level 10-26
- GET\$LEVEL -- Returns the interrupt level of highest priority for which an interrupt handler has started but has not yet finished processing 10-33

NUCLEUS SYSTEM CALLS

THE SYSTEM CALLS

CATALOG OBJECT

CATALOG\$OBJECT places an entry for an object in an object directory.

```
CALL RQ$CATALOG$OBJECT (job, object, name, except$ptr);
```

INPUT PARAMETERS

job	A WORD which, <ul style="list-style-type: none">• if zero, indicates that the object is to be cataloged in the object directory of the job to which the calling task belongs.• if not zero, contains the token for the job in whose object directory the object is to be cataloged.
object	A WORD containing a token for the object to be cataloged.
name	A POINTER to a STRING containing the ASCII name under which the object is to be cataloged. The name itself must not exceed 12 ASCII characters in length.

OUTPUT PARAMETER

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

DESCRIPTION

The CATALOG\$OBJECT system call places an entry for an object in the object directory of a specific job. The entry consists of both an ASCII name and the object. There may be several such entries for a single object in a directory, because the object may have several ASCII names. (However, in a given object directory, only one object may be cataloged under a given name.) If another task is waiting, via the LOOKUP\$OBJECT system call, for the object to be cataloged, that task is awakened when the entry is cataloged.

CONDITION CODES

E\$OK	No exceptional conditions.
-------	----------------------------

NUCLEUS SYSTEM CALLS

CATALOG OBJECT (continued)

CONDITION CODES (continued)

E\$CONTEXT	At least one of the following is true: <ul style="list-style-type: none">• The name being cataloged is already in the designated object directory.• The directory's maximum allowable size is 0.
E\$EXIST	Either the job parameter (which is not zero) or the object parameter is not a token for an existing object.
E\$LIMIT	The designated object directory is full.
E\$NOT\$CON- FIGURED	This system call is not part of the present configuration.
E\$PARAM	The first BYTE of the STRING pointed to by the name parameter contains a value greater than 12 or a value of 0.
E\$TYPE	The job parameter is a token for an object which is not a job.

NUCLEUS SYSTEM CALLS

CREATE JOB

CREATE\$JOB creates a job with a single task.

```
job = RQ$CREATE$JOB (directory$size, param$obj, pool$min,
pool$max, max$objects, max$tasks, max$priority,
except$handler, job$flags, task$priority, start$-
address, data$seg, stack$ptr, stack$size, task$flags,
except$ptr);
```

INPUT PARAMETERS

- directory\$size A WORD containing the maximum allowable number of entries in the created job's object directory. The value zero is permitted, for the case where no object directory is desired.
- param\$obj A WORD which,
- if zero, indicates that the new job has no parameter object.
 - if not zero, contains a token for the new job's parameter object.
- pool\$min A WORD which contains the minimum allowable size of the new job's pool, in 16 byte paragraphs. The pool\$min parameter is also the initial size of the new job's pool. If the stack\$ptr parameter has a base value of 0, pool\$min should be at least 32 plus the value of stack\$size in 16 byte paragraphs. Otherwise, pool\$min should be at least 32. If pool\$min is less than 32, an E\$PARAM exceptional condition occurs.
- pool\$max A WORD which contains the maximum allowable size of the new job's memory in 16 byte paragraphs. If pool\$max is zero or smaller than pool\$min, an E\$PARAM error occurs.
- max\$objects A WORD which,
- if not OFFFFH, contains the maximum number of objects, created by tasks in the new job, that can exist simultaneously.
 - if OFFFFH, indicates that there is no limit to the number of objects that tasks in the new job can create.

NUCLEUS SYSTEM CALLS

CREATE JOB (continued)
INPUT PARAMETERS (continued)

- max\$tasks** A WORD which,
- if not OFFFFH, contains the maximum number of tasks that can exist simultaneously in the new job.
 - if OFFFFH, indicates that there is no limit to the number of tasks that tasks in the new job can create.
- max\$priority** A BYTE which,
- if not zero, contains the maximum allowable priority of tasks in the new job. If max\$priority exceeds the maximum priority of the parent job, an E\$LIMIT error occurs.
 - if zero, indicates that the new job is to inherit the maximum priority attribute of the job to which the calling task belongs.
- except\$handler** A POINTER to a structure of the following form:
- ```

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

```

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

| <u>Value</u> | <u>When Control Passes To Exception Handler</u> |
|--------------|-------------------------------------------------|
| 0            | Never                                           |
| 1            | On programmer errors only                       |
| 2            | On environmental conditions only                |
| 3            | On all exceptional conditions                   |

NUCLEUS SYSTEM CALLS

CREATE JOB (continued)

INPUT PARAMETERS (continued)

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

| <u>bit</u> | <u>meaning</u>                                                                                                                                                                                                                                                                                                                                                                 |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 15-2       | reserved.                                                                                                                                                                                                                                                                                                                                                                      |
| 1          | <p>If 1, then whenever a task in the new job or any of the descendent jobs makes a system call, the Nucleus will check the parameters for validity.</p> <p>If 0, system calls by tasks in the new job will not be checked for validity. However, if any ancestor of the new job has been created with this bit set to 1, there will be parameter checking for the new job.</p> |
| 0          | reserved.                                                                                                                                                                                                                                                                                                                                                                      |

**task\$priority** A BYTE which,

- if not zero, contains the priority of the new job's initial task. If the task\$priority parameter is greater (numerically smaller) than the new job's maximum priority attribute, an E\$PARAM error occurs.
- if zero, indicates that the new job's initial task is to have a priority equal to the new job's maximum priority attribute.

**start\$address** A POINTER to the first instruction of the new job's initial task.

**data\$seg** A WORD which,

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

DB

## NUCLEUS SYSTEM CALLS

### Create Job (continued)

#### Input Parameters data segment (continued)

- if zero, indicates that the new job's initial task has no data segment.
- stack\$ptr            A POINTER which,
- if the base portion is not zero, points to the base of the user-provided stack of the new job's initial task.
  - if the base portion is zero, indicates that the Nucleus should allocate a stack segment to the new job's initial task. The length of the allocated segment is equal to the value of the stack\$size parameter.
- stack\$size            A WORD containing the size, in bytes, of the stack segment of the new job's initial task. Stack\$size must specify at least 16 bytes. The Nucleus increases specified values that are not multiples of 16 up to the next higher multiple of 16.
- Stack\$size should be at least 512 bytes if the new task is going to make system calls.
- task\$flags            A WORD reserved for future use. It should be set to 0.

#### OUTPUT PARAMETERS

- job                    A WORD containing a token for the new job.
- except\$ptr            A POINTER to a WORD to which the condition code for the call is to be returned.

#### DESCRIPTION

The CREATE\$JOB system call creates a job with an initializing task and returns a token for the job. The new job's parent is the calling task's job. The new job counts as one against the parent job's object limit. The new task counts as one against the new job's object and task limits. If a stack segment is created for the new task, it counts as one against the new job's object limit. The new job's resources come from the parent job, as described in the chapter on job management. In particular, the max\$task and max\$objects values are deducted from the creating job's maximum task and maximum objects attributes, respectively.

## NUCLEUS SYSTEM CALLS

## CREATE JOB (continued)

## CONDITION CODES

- E\$OK** No exceptional conditions.
- E\$CONTEXT** The job containing the calling task is in the process of being deleted.
- E\$EXIST** Param\$obj is not zero and is not a token for an existing object.
- E\$LIMIT** At least one of the following is true:
- pool\$min is larger than the available memory space in the memory pool of the calling task's job.
  - max\$objects is larger than the unused portion of the object allotment in the calling task's job.
  - max\$tasks is larger than the unused portion of the task allotment in the calling task's job.
  - max\$priority is larger than the maximum allowable task priority in the calling task's job.
  - the new job and tasks would exceed the object limit in the calling task's job.
  - the new task would exceed the object limit in the new job.
  - the new task would exceed the task limit in the calling task's job.
- E\$MEM** At least one of the following is true:
- the memory available to the new job is not sufficient to create a task.
  - the token part of the stack\$ptr parameter is zero, and the memory available to the new job is not sufficient to create a segment of the size indicated by the stack\$size parameter.

## NUCLEUS SYSTEM CALLS

CREATE JOB (continued)

CONDITION CODES

E\$PARAM

At least one of the following is true:

- pool\$min is less than 32.
- pool\$min is greater than pool\$max.
- task\$priority is greater (numerically smaller) than max\$priority.
- stack\$size is less than 16.

## NUCLEUS SYSTEM CALLS

## CREATE MAILBOX

CREATE\$MAILBOX creates a mailbox.

```
mailbox = RQ$CREATE$MAILBOX (mailbox$flags, except$ptr);
```

## INPUT PARAMETERS

mailbox\$flags A WORD containing information about the new mailbox. The low-order bit determines the queueing scheme for the new mailbox's task queue:

| <u>Value</u> | <u>Queueing Scheme</u> |
|--------------|------------------------|
| 0            | First-in-first-out     |
| 1            | Priority Based         |

The remaining bits in mailbox\$flags are reserved for future use and should be set to 0.

## OUTPUT PARAMETERS

mailbox A WORD containing a token for the new mailbox.

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

## DESCRIPTION

The CREATE\$MAILBOX system call creates a mailbox and returns a token for it. The new mailbox counts as one against the object limit of the calling task's job.

## CONDITION CODES

E\$OK No exception conditons.

E\$LIMIT The requested mailbox would exceed the job object limit.

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

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

## NUCLEUS SYSTEM CALLS

## CREATE SEGMENT

CREATE\$SEGMENT creates a segment.

```
segment = RQ$CREATE$SEGMENT (size, except$ptr);
```

## INPUT PARAMETER

|      |                                                                                                                                                                                                                                                                                                                                                                                  |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| size | A WORD which, <ul style="list-style-type: none"> <li>• if not zero, contains the size, in bytes, of the requested segment. If the size parameter is not a multiple of 16, it will be rounded up to the nearest higher multiple of 16 before the request is processed by the Nucleus.</li> <li>• if zero, indicates that the size of the request is 65536 (64K) bytes.</li> </ul> |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## OUTPUT PARAMETERS

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| segment     | A WORD which contains a token for the new segment.                        |
| except\$ptr | A POINTER to a WORD to which the condition code for the call is returned. |

## DESCRIPTION

The CREATE\$SEGMENT system call creates a segment and returns the token for it. The memory for the segment is taken from the free portion of the memory pool of the calling task's job, unless borrowing from the parent job is both necessary and possible. The new segment counts as one against the object limit of the calling task's job.

## CONDITION CODES

|                         |                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                                            |
| E\$LIMIT                | The requested segment would exceed the job object limit.                              |
| E\$MEM                  | The memory available to the calling task's job is not sufficient to create a segment. |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.                            |



## NUCLEUS SYSTEM CALLS

## CREATE SEMAPHORE

CREATE\$SEMAPHORE creates a semaphore.

```
semaphore = RQCREATE$SEMAPHORE (initial$value, max$value,
 semaphore$flags, except$ptr);
```

## INPUT PARAMETERS

**initial\$value** A WORD containing the initial number of units to be in the custody of the new semaphore.

**max\$value** A WORD containing the maximum number of units over which the new semaphore is to have custody at any given time. If max\$value is zero, an E\$PARAM error occurs.

**semaphore\$flags** A WORD containing information about the new semaphore. The low-order bit determines the queueing scheme for the new semaphore's task queue:

| <u>Value</u> | <u>Queueing Scheme</u> |
|--------------|------------------------|
| 0            | First-in-first-out     |
| 1            | Priority based         |

The remaining bits in semaphore\$flags are reserved for future use and should be set to 0.

## OUTPUT PARAMETERS

**semaphore** A WORD containing a token for the new semaphore.

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

## DESCRIPTION

The CREATE\$SEMAPHORE system call creates a semaphore and returns a token for it. The semaphore thus created counts as one against the object limit of the calling task's job.

## NUCLEUS SYSTEM CALLS

## CREATE SEMAPHORE (continued)

## CONDITION CODES

|                         |                                                                                                                                                                                                                   |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                                                                                                                                                                        |
| E\$LIMIT                | The requested semaphore would exceed the job object limit.                                                                                                                                                        |
| E\$MEM                  | The memory available to the calling task's job is not sufficient to create a semaphore.                                                                                                                           |
| E\$PARAM                | At least one of the following is true: <ul style="list-style-type: none"><li>• the initial\$value parameter is larger than the maximum\$value parameter or</li><li>• the maximum\$value parameter is 0.</li></ul> |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.                                                                                                                                                        |

## NUCLEUS SYSTEM CALLS

## CREATE TASK

CREATE\$TASK creates a task.

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

## INPUT PARAMETERS

|                |                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| priority       | A BYTE which, <ul style="list-style-type: none"><li>● if not zero, contains the priority of the new task. The priority parameter must not exceed the maximum allowable priority of the calling task's job. If it does, an E\$PARAM error occurs.</li><li>● if zero, indicates that the new task's priority is to equal the maximum allowable priority of the calling task's job.</li></ul> |
| start\$address | A POINTER to the first instruction of the new task.                                                                                                                                                                                                                                                                                                                                        |
| data\$seg      | A WORD which, <ul style="list-style-type: none"><li>● if not zero, contains the base address of the new task's data segment.</li><li>● if zero, indicates that the new task has no data segment.</li></ul>                                                                                                                                                                                 |
| stack\$ptr     | A POINTER which, <ul style="list-style-type: none"><li>● if the base portion is not zero, points to the base of the new task's stack.</li><li>● if the base portion is zero, indicates that the Nucleus should allocate a stack segment to the new task. The length of the allocated segment is equal to the value of the stack\$size parameter.</li></ul>                                 |
| stack\$size    | A WORD containing the size, in bytes, of the new task's stack segment. Stack\$size must specify at least 16 bytes. The Nucleus increases specified values that are not multiples of 16 up to the next higher multiple of 16.                                                                                                                                                               |

## NUCLEUS SYSTEM CALLS

## CREATE TASK (continued)

INPUT PARAMETERS  
stack\$size (continued)

Stack\$size should be at least 512 bytes if the new task is going to make system calls.

task\$flags            A WORD reserved for future use. It should be set to 0.

## OUTPUT PARAMETERS

task                    A WORD containing a token for the new task.

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

## DESCRIPTION

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

- priority: as specified in the call.
- execution state: ready.
- suspension depth: 0.
- containing job: the job which contains the calling task.
- exception handler: the exception handler of the containing job.
- exception mode: the exception mode of the containing job.

## CONDITION CODES

E\$OK                    No exceptional conditions.

E\$LIMIT                The new task would exceed the object limit or the task limit of the calling task's job.

## NUCLEUS SYSTEM CALLS

## CREATE TASK (continued)

## CONDITION CODES (continued)

E\$MEM

At least one of the following is the case:

- The memory available to the calling task's job is not sufficient to create a task.
- The base part of the stack\$ptr parameter is zero, and the memory available to the calling task's job is not sufficient to create a segment of the size indicated by the stack\$size parameter.

E\$NOT\$CON-  
FIGURED

This system call is not part of the present configuration.

E\$PARAM

At least one of the following is the case:

- The priority parameter is greater (numerically smaller) than the maximum allowable priority for tasks in the calling task's job.
- The stack\$size parameter is less than 16.

## NUCLEUS SYSTEM CALLS

## DELETE JOB

DELETE\$JOB deletes a job.

```
CALL RQ$DELETE$JOB (job, except$ptr);
```

## INPUT PARAMETER

**job** A WORD containing a token for the job to be deleted. A value of zero specifies the calling task's job.

## OUTPUT PARAMETERS

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

## DESCRIPTION

The DELETE\$JOB system call deletes from the system the specified job, as well as all objects created by tasks in it. Exceptions are that jobs, interrupt tasks, and extension objects (see the iRMX 86 System Programmer's Reference Manual) created by tasks in the target job must be deleted prior to the call to DELETE\$JOB. Information concerning the descendents of a job is obtained via the OFFSPRING system call. During deletion, all resources that the target job had borrowed from its parent are returned.

Deleting a job causes a credit of one toward the object total of the parent job. Also, the maximum tasks and maximum objects attributes of the deleted job are subtracted from the current tasks and current objects attributes, respectively, of the parent job.

## CONDITION CODES

**E\$OK** No exceptional conditions.

**E\$CONTEXT** There are undeleted jobs, interrupt tasks, or extension objects (see the iRMX 86 System Programmer's Reference Manual) which have been created by tasks in the target job.

**E\$EXIST** The job parameter is not a token for an existing object.

# NUCLEUS SYSTEM CALLS

## DELETE JOB (continued)

### CONDITION CODES (continued)

- E\$MEM**            The job to be deleted contains undeleted composite objects (see the iRMX 86 System Programmer's Reference Manual), and there is not sufficient memory for the Nucleus to send deletion messages to the appropriate deletion mailboxes.
- E\$NOT\$CON-  
FIGURED**        This system call is not part of the present configuration.
- E\$TYPE**           The job parameter is a token for an object that is not a job.

## NUCLEUS SYSTEM CALLS

## DELETE MAILBOX

DELETE\$MAILBOX deletes a mailbox.

```
CALL RQ$DELETE$MAILBOX (mailbox, except$ptr);
```

## INPUT PARAMETER

mailbox            A WORD containing a token for the mailbox to be deleted.

## OUTPUT PARAMETERS

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

## DESCRIPTION

The DELETE\$MAILBOX system call deletes the specified mailbox from the system. If any tasks are queued at the mailbox at the moment of deletion, they are awakened with an E\$EXIST exceptional condition. If there is a queue of object tokens at the moment of deletion, the queue is discarded. Deleting the mailbox counts as a credit of one toward the object total of the containing job.

## CONDITION CODES

E\$OK              No exceptional conditions.

E\$EXIST            The mailbox parameter is not a token for an existing object.

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

E\$TYPE            The mailbox parameter is a token for an object which is not a mailbox.



## NUCLEUS SYSTEM CALLS

## DELETE SEGMENT

DELETE\$SEGMENT deletes a segment.

```
CALL RQ$DELETE$SEGMENT (segment, except$ptr);
```

## INPUT PARAMETER

segment            A WORD containing a token for the segment  
                    that is to be deleted.

## OUTPUT PARAMETER

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

## DESCRIPTION

The DELETE\$SEGMENT system call returns the specified segment to the memory pool from which it was allocated. The deleted segment counts as a credit of one toward the object total of the containing job.

## CONDITION CODES

|                         |                                                                       |
|-------------------------|-----------------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                            |
| E\$EXIST                | The segment parameter is not a token for an existing object.          |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.            |
| E\$TYPE                 | The segment parameter is a token for an object that is not a segment. |

## NUCLEUS SYSTEM CALLS

## DELETE SEMAPHORE

DELETE\$SEMAPHORE deletes a semaphore.

```
CALL RQ$DELETE$SEMAPHORE (semaphore, except$ptr);
```

## INPUT PARAMETER

semaphore        A WORD containing a token for the semaphore that is to be deleted.

## OUTPUT PARAMETER

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

## DESCRIPTION

The DELETE\$SEMAPHORE system call deletes the specified semaphore. If there are tasks in the semaphore's queue at the moment of deletion, they are awakened with an E\$EXIST exceptional condition. The deleted semaphore counts as a credit of one toward the object total of the containing job.

## CONDITION CODES

E\$OK             No exceptional conditions.

E\$EXIST          The semaphore parameter is not a token for an existing object.

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

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

## NUCLEUS SYSTEM CALLS

## DELETE TASK

DELETE\$TASK deletes a task.

```
CALL RQ$DELETE$TASK (task, except$ptr);
```

## INPUT PARAMETER

|      |                                                                                                                                                                                             |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| task | A WORD which,                                                                                                                                                                               |
|      | <ul style="list-style-type: none"> <li>● if not zero, contains a token for the task that is to be deleted.</li> <li>● if zero, indicates that the calling task is to be deleted.</li> </ul> |

## OUTPUT PARAMETER

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| except\$ptr | A POINTER to a WORD to which the condition code for the call is to be returned. |
|-------------|---------------------------------------------------------------------------------|

## DESCRIPTION

The DELETE\$TASK system call deletes the specified task from the system and from any queues in which the task was waiting. Deleting the task counts as a credit of one toward the object total of the containing job. It also counts as a credit of one toward the containing job's task total. If the task's stack segment was allocated by the system, it also is deleted.

Interrupt tasks cannot be deleted by DELETE\$TASK; instead, interrupt tasks are deleted by RESET\$INTERRUPT.

## CONDITION CODES

|                         |                                                                  |
|-------------------------|------------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                       |
| E\$CONTEXT              | The task parameter is a token for an interrupt task.             |
| E\$EXIST                | The task parameter is not a token for an existing object.        |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.       |
| E\$TYPE                 | The task parameter is a token for an object which is not a task. |

## NUCLEUS SYSTEM CALLS

### DISABLE

DISABLE disables an interrupt level.

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

### INPUT PARAMETER

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

| <u>Bits</u> | <u>Value</u>              |
|-------------|---------------------------|
| 15-7        | 0                         |
| 6-4         | The interrupt level (0-7) |
| 3           | 1                         |
| 2-0         | 0                         |

### OUTPUT PARAMETER

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

### DESCRIPTION

The DISABLE system call disables the specified interrupt level. It has no effect on other levels. Level 2, which is reserved for the system clock, should not be disabled.

### CONDITION CODES

E\$OK            No exceptional conditions.

E\$CONTEXT      The level indicated by the level parameter is already disabled.

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

E\$PARAM        The level parameter is invalid.

## NUCLEUS SYSTEM CALLS

ENABLE

ENABLE enables an interrupt level.

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

## INPUT PARAMETER

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

| <u>Bits</u> | <u>Value</u>              |
|-------------|---------------------------|
| 15-7        | 0                         |
| 6-4         | the interrupt level (0-7) |
| 3           | 1                         |
| 2-0         | 0                         |

## OUTPUT PARAMETER

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

## DESCRIPTION

The ENABLE system call enables the specified interrupt level. The level must have an interrupt handler assigned to it.

## CONDITION CODES

E\$OK            No exceptional conditions.

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

- The level indicated by the level parameter is already disabled.
- There is not an interrupt handler assigned to the specified level.

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

E\$PARAM        The level parameter is invalid.

## NUCLEUS SYSTEM CALLS

## ENTER INTERRUPT

ENTER\$INTERRUPT is used by interrupt handlers to load a previously specified segment base address into the DS register.

```
CALL RQ$ENTER$INTERRUPT(level, except$ptr);
```

## INPUT PARAMETER

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

| <u>Bits</u> | <u>Value</u>              |
|-------------|---------------------------|
| 15-7        | 0                         |
| 6-4         | the interrupt level (0-7) |
| 3           | 1                         |
| 2-0         | 0                         |

## OUTPUT PARAMETER

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

## DESCRIPTION

ENTER\$INTERRUPT, on behalf of the calling interrupt handler, loads a base address value into the DS register. The value is what was specified when the interrupt handler was set up by an earlier call to SET\$INTERRUPT.

One purpose of loading a new value into the DS register is to protect the contents of the interrupted task's data segment. Another purpose, if the handler is going to call an interrupt task, is that ENTER\$INTERRUPT enables the handler to place data in the iAPX 86 data segment that will be used by the interrupt task. This provides a mechanism for the interrupt handler to pass data to the interrupt task.

NUCLEUS SYSTEM CALLS

ENTER INTERRUPT

ENTER INTERRUPT (continued)

CONDITION CODES

- E\$OK No exceptional conditions.
- E\$CONTEXT No value had previously been specified in the call to SET\$INTERRUPT.
- E\$NOT\$CON-FIGURED This system call is not included in the present configuration.
- E\$PARAM The level parameter is invalid.

SYSTEM CALLS

## NUCLEUS SYSTEM CALLS

## EXIT INTERRUPT

EXIT\$INTERRUPT is used by interrupt handlers when they don't call interrupt tasks; this call sends an end-of-interrupt signal to hardware.

CALL RQ\$EXIT\$INTERRUPT (level, except\$ptr);

## INPUT PARAMETER

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

| <u>Bits</u> | <u>Value</u>              |
|-------------|---------------------------|
| 15-7        | 0                         |
| 6-4         | the interrupt level (0-7) |
| 3           | 1                         |
| 2-0         | 0                         |

## OUTPUT PARAMETER

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

## DESCRIPTION

The EXIT\$INTERRUPT system call sends an end-of-interrupt signal to hardware. This sets the stage for re-enabling interrupts. The re-enabling actually occurs when control passes from the interrupt handler to an application task.

## CONDITION CODES

|                         |                                                            |
|-------------------------|------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                 |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration. |
| E\$PARAM                | The level parameter is invalid.                            |



## NUCLEUS SYSTEM CALLS

## GET EXCEPTION HANDLER

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

```
CALL RQGETEXCEPTION$HANDLER (exception$info$ptr, except$ptr);
```

## OUTPUT PARAMETERS

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

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

where, after the call,

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

| <u>Value</u> | <u>When to Pass Control<br/>to Exception Handler</u> |
|--------------|------------------------------------------------------|
| 0            | Never                                                |
| 1            | On programmer errors only                            |
| 2            | On environmental conditions only                     |
| 3            | On all exceptional conditons                         |

except\$ptr

A POINTER to a WORD to which the condition code for the call is to be returned.

## GET EXCEPTION HANDLER (continued)

## DESCRIPTION

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

## CONDITION CODE

|                         |                                                            |
|-------------------------|------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                 |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration. |

NUCLEUS SYSTEM CALLS

GET LEVEL

GET\$LEVEL returns the number of the level of the interrupt being serviced.

```
level = RQGETLEVEL (except$ptr);
```

INPUT PARAMETERS

none

OUTPUT PARAMETERS

level            A WORD whose value is interpreted as follows (bit 15 is the high-order bit):

| <u>Bits</u> | <u>Value/Interpretation</u>                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                             |   |                                                           |   |   |                                                             |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|---|-----------------------------------------------------------|---|---|-------------------------------------------------------------|
| 15-8        | undefined                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                             |   |                                                           |   |   |                                                             |
| 7           | <table style="border: none; display: inline-table; vertical-align: middle;"> <tr> <td style="font-size: 3em; padding-right: 5px;">{</td> <td style="padding: 0 5px;">0</td> <td style="padding: 0 5px;">some level is being serviced and bits 6-4 are significant</td> </tr> <tr> <td style="font-size: 3em; padding-right: 5px;">}</td> <td style="padding: 0 5px;">1</td> <td style="padding: 0 5px;">no level is being serviced and bits 6-4 are not significant</td> </tr> </table> | {                                                           | 0 | some level is being serviced and bits 6-4 are significant | } | 1 | no level is being serviced and bits 6-4 are not significant |
| {           | 0                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | some level is being serviced and bits 6-4 are significant   |   |                                                           |   |   |                                                             |
| }           | 1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | no level is being serviced and bits 6-4 are not significant |   |                                                           |   |   |                                                             |
| 6-4         | an interrupt level (0-7)                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                             |   |                                                           |   |   |                                                             |
| 3-0         | undefined                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                             |   |                                                           |   |   |                                                             |

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

DESCRIPTION

The GET\$LEVEL system call returns to the calling task the highest (numerically lowest) level which an interrupt handler has started servicing but has not yet finished. To strip away unwanted one bits, logically AND the returned value with OOF0H.

CONDITION CODES

- E\$OK            No exceptional conditions.
- E\$NOT\$CON-  
FIGURED        This system call is not part of the present configuration.

NUCLEUS SYSTEM CALLS

GET POOL ATTRIB

GET\$POOL\$ATTRIB returns information about the memory pool of the calling task's job.

```
CALL RQGETPOOL$ATTRIB (attrib$ptr, except$ptr);
```

INPUT PARAMETER

attrib\$ptr A POINTER to a data structure of the following form:

```
STRUCTURE(
 POOL$MAX WORD,
 POOL$MIN WORD,
 INITIAL$SIZE WORD,
 ALLOCATED WORD,
 AVAILABLE WORD);
```

where, after the call,

- pool\$max contains the maximum allowable size of the memory pool of the calling task's job.
- pool\$min contains the minimum allowable size of the memory pool of the calling task's job.
- initial\$size contains the original value of the pool\$min attribute.
- allocated contains the number of bytes currently allocated from the memory pool of the calling task's job.
- available contains the number of bytes currently available in the memory pool of the calling task's job.

OUTPUT PARAMETER

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

DESCRIPTION

The GET\$POOL\$ATTRIB system call returns information regarding the memory pool of the calling task's job. The data returned comprises the allocated and available portions of the pool, as well as its initial, minimum, and maximum sizes.

NUCLEUS SYSTEM CALLS

GET POOL ATTRIB (continued)

CONDITION CODE

E\$OK

No exceptional conditions.

E\$NOT\$CON-  
FIGURED

This system call is not part of the present  
configuration.

NUCLEUS SYSTEM CALLS

GET PRIORITY

GET\$PRIORITY returns the priority of a task.

priority = RQ\$GET\$PRIORITY (task, except\$ptr);

INPUT PARAMETER

|      |                                                                                                                                                                                                                       |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| task | A WORD which,                                                                                                                                                                                                         |
|      | <ul style="list-style-type: none"> <li>• if not zero, contains a token for the task whose priority is being requested.</li> <li>• if zero, indicates that the calling task is asking for its own priority.</li> </ul> |

OUTPUT PARAMETERS

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| priority    | A BYTE containing the priority of the task indicated by the task parameter.     |
| except\$ptr | A POINTER to a WORD to which the condition code for the call is to be returned. |

DESCRIPTION

The GET\$PRIORITY system call returns the priority of the specified task.

CONDITION CODES

|                         |                                                                 |
|-------------------------|-----------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                      |
| E\$EXIST                | The task parameter is not a token for an existing object.       |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.      |
| E\$TYPE                 | The task parameter is a token for an object that is not a task. |

NUCLEUS SYSTEM CALLS

GET SIZE

GET\$SIZE returns the size, in bytes, of a segment.

```
size = RQGETSIZE (segment, except$ptr);
```

INPUT PARAMETER

segment            A WORD containing a token for a segment.

OUTPUT PARAMETERS

size                A WORD which,

- if not zero, contains the size, in bytes, of the segment indicated by the segment parameter.
- if zero, indicates that the size of the segment is 65536 (64K) bytes.

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

DESCRIPTION

The GET\$SIZE system call returns the size, in bytes, of a segment.

CONDITION CODES

- E\$OK                No exceptional conditons.
- E\$EXIST            The segment parameter is not a token for an existing object.
- E\$NOT\$CON-  
FIGURED            This system call is not part of the present configuration.
- E\$TYPE             The segment parameter is a token for an object that is not a segment.

SYSTEM CALLS

NUCLEUS SYSTEM CALLS

GET TASK TOKENS

GET\$TASK\$TOKENS returns the token requested by the calling task.

```
token = RQGETTASK$TOKENS (selection, except$ptr);
```

INPUT PARAMETER

selection                    A BYTE containing the request, encoded as follows:

| <u>Value</u> | <u>Object for which a Token is Requested</u>    |
|--------------|-------------------------------------------------|
| 0            | The calling task.                               |
| 1            | The calling task's job.                         |
| 2            | The parameter object of the calling task's job. |
| 3            | The root job.                                   |

OUTPUT PARAMETERS

token                        A WORD containing the requested token.

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

DESCRIPTION

The GET\$TASK\$TOKENS system call returns a token for either the calling task, the calling task's job, the calling task's parameter object, or the root job, depending on the encoded request.

CONDITION CODES

E\$OK                         No exceptional conditions.

E\$PARAM                      The selection parameter is greater than 3.



## NUCLEUS SYSTEM CALLS

## GET TYPE

GET\$TYPE returns the encoded type of an object.

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

## INPUT PARAMETER

object            A WORD containing the token for an object.

## OUTPUT PARAMETERS

type\$code        A WORD which contains the encoded type of the specified object. The types are encoded as follows:

| <u>Value</u> | <u>Type</u> |
|--------------|-------------|
| 1            | job         |
| 2            | task        |
| 3            | mailbox     |
| 4            | semaphore   |
| 6            | segment     |

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

## DESCRIPTION

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

## CONDITION CODES

E\$OK             No exceptional conditions.

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

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

## NUCLEUS SYSTEM CALLS

## LOOKUP OBJECT

LOOKUP\$OBJECT returns a token for a cataloged object.

```
object = RQ$LOOKUP$OBJECT (job, name, time$limit, except$ptr);
```

## INPUT PARAMETERS

- job**                    A WORD which,
- if not zero, contains a token for the job whose object directory is to be searched.
  - if zero, indicates that the object directory to be searched is that of the calling task's job.
- name**                    A POINTER to a STRING which contains the ASCII name under which the object is cataloged. During the lookup operation, upper and lower case letters are treated as being different.
- time\$limit**            A WORD which,
- if zero, indicates that the calling task is not willing to wait.
  - if OFFFFH, indicates that the task will wait as long as is necessary.
  - if between 0 and OFFFFH, indicates that the task is willing to wait only that many 1/100 second time units.

## OUTPUT PARAMETERS

- object**                    A WORD containing the requested token.
- except\$ptr**            A POINTER to a WORD to which the condition code for the call is to be returned.

## DESCRIPTION

The LOOKUP\$OBJECT system call returns the token for the specified object after searching for its ASCII name in the specified object directory. Because it is possible that the object is not cataloged at the time of the call, the calling task has the option of waiting, either indefinitely or for a specific period of time, for another task to catalog the object.

## NUCLEUS SYSTEM CALLS

## LOOKUP OBJECT (continued)

## CONDITION CODES

|                         |                                                                                                                                                                                                                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                                                                                                                                                                                                                                                                    |
| E\$CONTEXT              | The specified job has an object directory of size 0.                                                                                                                                                                                                                                                          |
| E\$EXIST                | The job parameter (which is not zero) is not a token for an existing object.                                                                                                                                                                                                                                  |
| E\$LIMIT                | The specified object directory is full.                                                                                                                                                                                                                                                                       |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.                                                                                                                                                                                                                                                    |
| E\$PARAM                | The first byte of the string pointed to by the name parameter contains a value greater than 12 or equal to zero.                                                                                                                                                                                              |
| E\$TIME                 | Either <ul style="list-style-type: none"><li>● the calling task indicated its willingness to wait a certain amount of time, then waited without satisfaction or</li><li>● the task was not willing to wait, and the entry indicated by the name parameter is not in the specified object directory.</li></ul> |
| E\$TYPE                 | The job parameter is a token for an object that is not a job.                                                                                                                                                                                                                                                 |

## NUCLEUS SYSTEM CALLS

### OFFSPRING

OFFSPRING returns a token for each child (job) of a job.

```
token$list = RQ$OFFSPRING (job, except$ptr);
```

### INPUT PARAMETER

**job** A WORD containing a token for the job whose offspring are desired. A value of zero specifies the calling task's job.

### OUTPUT PARAMETER

**token\$list** A WORD which,

- if not zero, contains a token for a segment. The first word in the segment contains the number of words in the remainder of the segment. Subsequent words contain the tokens for jobs which are the children of the specified job.
- if zero, indicates that the specified job has no children.

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

### DESCRIPTION

The OFFSPRING system call returns the token for a segment. The segment contains a token for each child of the specified job. By repeated use of this call, tokens can be obtained for all descendents of a job; this information is needed by a task which is attempting to delete a job that has child jobs.

### CONDITION CODES

**E\$OK** No exceptional conditions.

**E\$EXIST** The job parameter is not a token for an existing object.

**E\$LIMIT** The required segment, if allocated, would exceed the job object limit.

**E\$MEM** There is not sufficient memory available to create the required segment.

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

**E\$TYPE** The job parameter is a token for an object that is not a job.

## NUCLEUS SYSTEM CALLS

## RECEIVE MESSAGE

RECEIVE\$MESSAGE delivers the calling task to a mailbox, where it waits for an object token to be returned.

```
object = RQ$RECEIVE$MESSAGE (mailbox, time$limit,
 response$ptr, except$ptr);
```

## INPUT PARAMETERS

- mailbox            A WORD containing a token for the mailbox at which the calling task expects to receive an object token.
- time\$limit        A WORD which,
- if zero, indicates that the calling task is not willing to wait.
  - if OFFFFH, indicates that the task will wait as long as is necessary.
  - if between 0 and OFFFFH, indicates that the task is willing to wait only that many 1/100 second time units.

## OUTPUT PARAMETERS

- object            A WORD containing the token for the object being received.
- response\$ptr     A POINTER to a WORD in which the system returns a value. The returned word,
- if not zero, contains a token for the exchange to which the receiving task is to send a response.
  - if zero, indicates that no response has been requested by the sending task.

## CAUTION

Response\$ptr points to a location for the sending task to use. If you specify a constant value for response\$ptr, be careful to ensure that the value does not conflict with system requirements. In particular, a value of zero for response\$ptr will normally cause unpredictable system behavior.

## NUCLEUS SYSTEM CALLS

RECEIVE MESSAGE (continued)

OUTPUT PARAMETERS (continued)

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| except\$ptr | A POINTER to a WORD to which the condition code for the call is to be returned. |
|-------------|---------------------------------------------------------------------------------|

## DESCRIPTION

The RECEIVE\$MESSAGE system call causes the calling task either to get the token for an object or to wait for the token in the task queue of the specified mailbox. If the object queue at the mailbox is not empty, then the calling task immediately gets the token at the head of the queue and remains ready. Otherwise, the calling task goes into the task queue of the mailbox and goes to sleep, unless the task is not willing to wait. In the latter case, or if the task's waiting period elapses without a token arriving, the task is awakened with an E\$TIME exceptional condition.

If the sending task needs a response from the receiving task, a token for the requested response exchange is returned in the word to which the response\$ptr parameter is pointing. The nature of the response must be agreed upon by the writers of the two tasks.

## CONDITION CODES

|                         |                                                                                                                                                                                                                                                                        |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                                                                                                                                                                                                                             |
| E\$EXIST                | The mailbox parameter is not a token for an existing object.                                                                                                                                                                                                           |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.                                                                                                                                                                                                             |
| E\$TIME                 | Either <ul style="list-style-type: none"><li>● the calling task was not willing to wait and there was not a token available, or</li><li>● the task waited in the task queue and its designated waiting period elapsed before the task got the desired token.</li></ul> |
| E\$TYPE                 | The mailbox parameter is a token for an object that is not a mailbox.                                                                                                                                                                                                  |

## NUCLEUS SYSTEM CALLS

## RECEIVE UNITS

RECEIVE\$UNITS delivers the calling task to a semaphore, where it waits for units.

```
value = RQ$RECEIVE$UNITS (semaphore, units, time$limit,
 except$ptr);
```

## INPUT PARAMETERS

- |             |                                                                                                                                                                                                                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| semaphore   | A WORD containing a token for the semaphore from which the calling task hopes to receive units.                                                                                                                                                                                                                                            |
| units       | A WORD containing the number of units that the calling task is requesting.                                                                                                                                                                                                                                                                 |
| time\$limit | A WORD which, <ul style="list-style-type: none"> <li>• if zero, indicates that the calling task is not willing to wait.</li> <li>• if OFFFFH, indicates that the task will wait as long as is necessary.</li> <li>• if between 0 and OFFFFH, indicates that the task is willing to wait only that many 1/100 second time units.</li> </ul> |

## OUTPUT PARAMETERS

- |             |                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------|
| value       | A WORD containing the number of units remaining in the custody of the semaphore after the calling task's request is satisfied. |
| except\$ptr | A POINTER to a WORD to which the condition code for the call is to be returned.                                                |

## DESCRIPTION

The RECEIVE\$UNITS system call causes the calling task either to get the units that it is requesting or to wait for them in the semaphore's task queue. If the units are available and the task is at the front of the queue, then the task receives them and remains ready. Otherwise, the task is placed in the semaphore's task queue and goes to sleep, unless the task is not willing to wait. In the latter case, or if the task's waiting period elapses before the requested units are available, the task is awakened with an E\$TIME exceptional condition.

## NUCLEUS SYSTEM CALLS

## RECEIVE UNITS (continued)

## CONDITION CODES

|                         |                                                                                                                                                                                                                                                                                  |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                                                                                                                                                                                                                                       |
| E\$EXIST                | The semaphore parameter is not a token for an existing object.                                                                                                                                                                                                                   |
| E\$LIMIT                | The units parameter is greater than the maximum value that had been specified for the semaphore when it was created.                                                                                                                                                             |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.                                                                                                                                                                                                                       |
| E\$TIME                 | Either <ul style="list-style-type: none"><li>● the calling task was not willing to wait and the requested units were not available or</li><li>● the task waited in the task queue and its designated waiting period elapsed before the requested units were available.</li></ul> |
| E\$TYPE                 | The semaphore parameter is a token for an object that is not a semaphore.                                                                                                                                                                                                        |



NUCLEUS SYSTEM CALLS

RESET INTERRUPT

RESET\$INTERRUPT cancels the assignment of an interrupt handler to a level.

```
CALL RQ$RESET$INTERRUPT (level, except$ptr);
```

INPUT PARAMETER

level            A WORD containing an interrupt level which is encoded as follows (bit 15 is the high-order bit):

| <u>Bits</u> | <u>Value</u>              |
|-------------|---------------------------|
| 15-7        | 0                         |
| 6-4         | the interrupt level (0-7) |
| 3           | 1                         |
| 2-0         | 0                         |

OUTPUT PARAMETER

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

DESCRIPTION

The RESET\$INTERRUPT system call cancels the assignment of the current interrupt handler to the specified interrupt level. If an interrupt task had also been assigned to the level, the interrupt task is deleted. RESET\$INTERRUPT also disables the level.

Level 2 should not be reset and is considered invalid.

CONDITION CODES

- E\$OK            No exceptional conditions.
- E\$CONTEXT      There is not an interrupt handler assigned to the specified level.
- E\$NOT\$CON-  
FIGURED        This system call is not part of the present configuration.
- E\$PARAM        The level parameter is invalid.

## NUCLEUS SYSTEM CALLS

## RESUME TASK

RESUME\$TASK decreases by one the suspension depth of a task.

```
CALL RQ$RESUME$TASK (task, except$ptr);
```

## INPUT PARAMETER

**task**                    A WORD containing a token for the task whose suspension depth is to be decremented.

## OUTPUT PARAMETER

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

## DESCRIPTION

The RESUME\$TASK system call decreases by one the suspension depth of the specified non-interrupt task. The task should be in either the suspended or asleep-suspended state, so its suspension depth should be at least one. If the suspension depth is still positive after being decremented, the state of the task is not changed. If the depth becomes zero, and the task is in the suspended state, then it is placed in the ready state. If the depth becomes zero, and the task is in the asleep-suspended state, then it is placed in the asleep state.

## CONDITION CODES

**E\$OK**                    No exceptional conditions.

**E\$CONTEXT**            The task indicated by the task parameter is an interrupt task.

**E\$EXIST**                The task parameter is not a token for an existing object.

**E\$STATE**                The task indicated by the task parameter was not suspended when the call was made.

**E\$TYPE**                The task parameter is a token for an object that is not a task.

## NUCLEUS SYSTEM CALLS

## SEND MESSAGE

SEND\$MESSAGE sends an object token to a mailbox.

```
CALL RQ$SEND$MESSAGE (mailbox, object, response, except$ptr);
```

## INPUT PARAMETERS

|          |                                                                                                                                                                                                          |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mailbox  | A WORD containing a token for the mailbox to which an object token is to be sent.                                                                                                                        |
| object   | A WORD containing an object token which is to be sent.                                                                                                                                                   |
| response | A WORD which, <ul style="list-style-type: none"><li>• if not zero, contains a token for the desired response mailbox or semaphore.</li><li>• if zero, indicates that no response is requested.</li></ul> |

## OUTPUT PARAMETER

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| except\$ptr | A POINTER to a WORD to which the condition code for the call is to be returned. |
|-------------|---------------------------------------------------------------------------------|

## DESCRIPTION

The SEND\$MESSAGE system call sends the specified object token to the specified mailbox. If there are tasks in the task queue at that mailbox, the task at the head of the queue is awakened and is given the token. Otherwise, the object token is placed at the tail of the object queue of the mailbox. The sending task has the option of specifying a mailbox or semaphore at which it will wait for a response from the task that receives the object. The nature of the response must be agreed upon by the writers of the two tasks.

## CONDITION CODES

|          |                                                                                                               |
|----------|---------------------------------------------------------------------------------------------------------------|
| E\$OK    | No exceptional conditions.                                                                                    |
| E\$EXIST | One or more of the input parameters is not a token for an existing object.                                    |
| E\$MEM   | There is not sufficient memory for the Nucleus to do the housekeeping that supports a send message operation. |

## NUCLEUS SYSTEM CALLS

SEND MESSAGE (continued)

CONDITION CODES (continued)

E\$NOT\$CON-  
FIGURED

This system call is not part of the present configuration.

E\$TYPE

Either

- the mailbox parameter is a token for an object that is not a mailbox or
- the response parameter is a token for an object that is neither a mailbox nor a semaphore.

# NUCLEUS SYSTEM CALLS

## SEND UNITS

SEND\$UNITS sends units to a semaphore.

```
CALL RQ$SEND$UNITS (semaphore, units, except$ptr);
```

### INPUT PARAMETERS

- semaphore      A WORD containing a token for the semaphore to which the units are to be sent.
- units            A WORD containing the number of units to be sent.

### OUTPUT PARAMETER

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

### DESCRIPTION

The SEND\$UNITS system call sends the specified number of units to the specified semaphore. If the transmission would cause the semaphore's supply of units to exceed its maximum allowable supply, then an E\$LIMIT exceptional condition occurs. Otherwise, the transmission is successful and the Nucleus attempts to satisfy the requests of the tasks in the semaphore's task queue, beginning at the head of the queue.

### CONDITION CODES

- E\$OK            No exceptional conditons.
- E\$EXIST        The semaphore parameter is not a token for an existing object.
- E\$LIMIT        The number of units that the calling task is trying to send would cause the semaphore's supply of units to exceed its maximum allowable supply.
- E\$NOT\$CON-  
FIGURED        This system call is not part of the present configuration.
- E\$TYPE        The semaphore parameter is a token for an object that is not a semaphore.

NUCLEUS SYSTEM CALLS

SET EXCEPTION HANDLER

SET\$EXCEPTION\$HANDLER assigns an exception handler to the calling task.

```
CALL RQSETEXCEPTION$HANDLER (exception$info$ptr, except$ptr);
```

INPUT PARAMETER

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

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

where

- exception\$handler\$offset contains the offset of the first instruction of the exception handler.
- exception\$handler\$base contains a token for the segment containing the first instruction of the exception handler.
- exception\$mode contains an encoded indication of the calling task's intended exception mode. The value is interpreted as follows:

| <u>Value</u> | <u>When to Pass Control<br/>To Exception Handler</u> |
|--------------|------------------------------------------------------|
| 0            | Never                                                |
| 1            | On programmer errors only                            |
| 2            | On environmental conditions only                     |
| 3            | On all exceptional conditions                        |

OUTPUT PARAMETER

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

SET EXCEPTION HANDLER (continued)

DESCRIPTION

The SET\$EXCEPTION\$HANDLER system call enables a task to set its exception handler and exception mode attributes. If you want to designate the Debugger as the exception handler to interactively examine system objects and lists, the following code sets up the needed structure in PL/M-86:

```

DECLARE X STRUCTURE (OFFSET WORD,
 BASE WORD,
 MODE BYTE);

DECLARE Y POINTER AT (@X);

DECLARE EXCEPTION WORD;

Y = @RQDEBUGGEREX;
X.MODE = ZEROONETWOORTHREE;
CALL RQSETEXCEPTION$HANDLER (@X, @EXCEPTION);

```

CONDITION CODES

- |                         |                                                            |
|-------------------------|------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                 |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration. |
| E\$PARAM                | The exception\$mode parameter is greater than 3.           |

## NUCLEUS SYSTEM CALLS

## SET INTERRUPT

SET\$INTERRUPT assigns an interrupt handler to an interrupt level and, optionally, makes the calling task the interrupt task for the level.

```
CALL RQSETINTERRUPT (level, interrupt$task$flag,
 interrupt$handler, interrupt$handler$ds, except$ptr);
```

## INPUT PARAMETERS

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

| <u>Bits</u> | <u>Value</u>              |
|-------------|---------------------------|
| 15-7        | 0                         |
| 6-4         | the interrupt level (0-7) |
| 3           | 1                         |
| 2-0         | 0                         |

**interrupt\$task\$flag** A BYTE which,

- if one, indicates that the calling task is to be the interrupt task that will be invoked by the interrupt handler being set. The priority of the calling task is set by the Nucleus. The priority is derived from the level, according to the following table:

| <u>Level</u> | <u>Priority</u> |
|--------------|-----------------|
| 0            | 18              |
| 1            | 34              |
| 2            | 50              |
| 3            | 66              |
| 4            | 82              |
| 5            | 98              |
| 6            | 114             |
| 7            | 130             |

Be certain that priorities set in this manner do not violate the max\$priority attribute of the containing job.



## NUCLEUS SYSTEM CALLS

## SET INTERRUPT (continued)

- if zero, indicates that no interrupt task is to be associated with the special level and that the new interrupt handler will not call SIGNAL INTERRUPT.
- if greater than one, causes an E\$PARAM exceptional condition.

interrupt\$handler

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

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

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

interrupt\$handler\$ds A WORD which,

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

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

NUCLEUS SYSTEM CALLS

SET INTERRUPT(continued)

INPUT PARAMETERS

interrupt\$handler\$ds (continued)

```

DECLARE BEGIN WORD; /* A DUMMY
 VARIABLE WHICH
 IS THE FIRST
 DECLARED
 VARIABLE */

DECLARE DATA$PTR POINTER;

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

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

DS$BASE = DATA$ADDRESS.BASE;

CALL RQSETINTERRUPT (... ,DS$BASE);

```

- if zero, indicates that the interrupt handler will use the data segment of the interrupted task and may not call ENTER\$INTERRUPT.

OUTPUT PARAMETER

except\$ptr

A POINTER to a WORD to which the condition code for the call is to be returned.

## NUCLEUS SYSTEM CALLS

## SET INTERRUPT (continued)

## DESCRIPTION

The SET\$INTERRUPT system call is used to inform the Nucleus that the specified interrupt handler is to service interrupts which come in at the specified level. In a call to SET\$INTERRUPT, a task must indicate whether the interrupt handler will invoke an interrupt task and whether the interrupt handler has its own data segment. If there is to be an interrupt task, the calling task is that interrupt task. If there is no interrupt task, SET\$INTERRUPT also enables the specified level, which must be disabled at the time of the call.

## CONDITION CODES

|                         |                                                                                                                                                                                                                               |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ESOK                    | No exceptional conditions.                                                                                                                                                                                                    |
| E\$CONTEXT              | Either <ul style="list-style-type: none"><li>● the specified level already has an interrupt handler assigned to it or</li><li>● the job containing the calling task is in the process of being deleted.</li></ul>             |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.                                                                                                                                                                    |
| E\$PARAM                | Either <ul style="list-style-type: none"><li>● the level parameter is invalid or would cause the task to have a priority not allowed by its job.</li><li>● the interrupt\$task\$flag parameter is greater than one.</li></ul> |

## SET POOL MINIMUM

SET\$POOL\$MINIMUM sets a job's pool\$min attribute.

```
CALL RQSETPOOL$MINIMUM (new$min, except$ptr);
```

## INPUT PARAMETER

new\$min            A WORD which,

- if OFFFFH, indicates that the pool\$min attribute of the calling task's job is to be set equal to that job's pool\$max attribute.
- if less than OFFFFH, contains the new value of the pool\$min attribute of the calling task's job. This new value must not exceed that job's pool\$max attribute.

## OUTPUT PARAMETER

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

## DESCRIPTION

The SET\$POOL\$MINIMUM system call sets the pool\$min attribute of the calling task's job. The new value must not exceed that job's pool\$max attribute. When the pool\$min attribute is made larger than the current pool size, the pool is not enlarged until the additional memory is needed.

## CONDITION CODES

E\$OK              No exceptional conditions.

E\$LIMIT           The new\$min parameter is not OFFFFH, yet is greater than the pool\$max attribute of the calling task's job.

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

NUCLEUS SYSTEM CALLS

SIGNAL INTERRUPT

SIGNAL\$INTERRUPT is used by an interrupt handler to activate an interrupt task.

CALL RQ\$SIGNAL\$INTERRUPT (level, except\$ptr);

INPUT PARAMETER

level                    A WORD containing an interrupt level which is encoded as follows (bit 15 is the high-order bit):

| <u>Bits</u> | <u>Value</u>              |
|-------------|---------------------------|
| 15-7        | 0                         |
| 6-4         | the interrupt level (0-7) |
| 3           | 1                         |
| 2-0         | 0                         |

OUTPUT PARAMETER

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

DESCRIPTION

An interrupt handler uses SIGNAL\$INTERRUPT to start up its associated interrupt task. The interrupt task runs in its own environment with higher level interrupts enabled, whereas the interrupt handler runs in the environment of the interrupted task with all interrupts disabled.

CONDITION CODES

- E\$OK                    No exceptional conditions.
- E\$CONTEXT            There is not an interrupt task assigned to the specified level.
- E\$NOT\$CON-  
FIGURED                This system call is not part of the present configuration.
- E\$PARAM                The level parameter is invalid.

## NUCLEUS SYSTEM CALLS

## SLEEP

SLEEP puts the calling task to sleep.

```
CALL RQ$SLEEP (time$limit, except$ptr);
```

## INPUT PARAMETER

- time\$limit**      A WORD which,
- if not zero and not OFFFFH, causes the calling task to go to sleep for that many 1/100 second time units, after which it will be awakened.
  - if zero, causes the calling task to be placed on the list of ready tasks, immediately behind all tasks of the same priority. If there are no such tasks, there is no effect.
  - if OFFFFH, is invalid.

## OUTPUT PARAMETER

- except\$ptr**      A POINTER to a WORD to which the condition code for the call is to be returned.

## DESCRIPTION

The SLEEP system call has two uses. One use places the calling task in the asleep state for a specific amount of time. The other use allows the calling task to defer to the other ready tasks with the same priority. When a task defers in this way it is placed on the list of ready tasks, immediately behind those other tasks of equal priority.

## CONDITION CODES

- E\$OK**              No exceptional conditions.
- E\$NOT\$CON-  
FIGURED**          This system call is not part of the present configuration.
- E\$PARAM**          The time\$limit parameter contains the invalid value OFFFFH.

## NUCLEUS SYSTEM CALLS

## SUSPEND TASK

SUSPEND\$TASK increases by one the suspension depth of a task.

```
CALL RQ$SUSPEND$TASK (task, except$ptr);
```

## INPUT PARAMETER

|      |                                                                                                                                                                                                                                  |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| task | A WORD which, <ul style="list-style-type: none"><li>● if not zero, contains a token for the task whose suspension depth is to be incremented.</li><li>● if zero, indicates that the calling task is suspending itself.</li></ul> |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## OUTPUT PARAMETER

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| except\$ptr | A POINTER to a WORD to which the condition code for the call is to be returned. |
|-------------|---------------------------------------------------------------------------------|

## DESCRIPTIONS

The SUSPEND\$TASK system call increases by one the suspension depth of the specified task. If the task is already in either the suspended or asleep-suspended state, its state is not changed. If the task is in the ready or running state, it enters the suspended state. If the task is in the asleep state, it enters the asleep-suspended state.

SUSPEND\$TASK cannot be used to suspend interrupt tasks.

## CONDITION CODES

|            |                                                                               |
|------------|-------------------------------------------------------------------------------|
| E\$OK      | No exceptional conditions.                                                    |
| E\$CONTEXT | The task indicated by the task parameter is an interrupt task.                |
| E\$EXIST   | The task parameter is not a token for an existing object.                     |
| E\$LIMIT   | The suspension depth for the specified task is already at the maximum of 255. |
| E\$TYPE    | The task parameter is a token for an object that is not a task.               |

NUCLEUS SYSTEM CALLS

UNCATALOG OBJECT

UNCATALOG\$OBJECT removes an entry for an object from an object directory.

```
CALL RQ$UNCATALOG$OBJECT (job, name, except$ptr);
```

INPUT PARAMETERS

- |      |                                                                                                                                                                                                                                                                                            |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| job  | A WORD which, <ul style="list-style-type: none"> <li>● if not zero, is a token for the job from whose object directory the specified entry is to be deleted.</li> <li>● if zero, indicates that the entry is to be deleted from the object directory of the calling task's job.</li> </ul> |
| name | A POINTER to a STRING containing the ASCII name of the object whose entry is to be deleted.                                                                                                                                                                                                |

OUTPUT PARAMETER

- |             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| except\$ptr | A POINTER to a WORD to which the condition code for the call is to be returned. |
|-------------|---------------------------------------------------------------------------------|

DESCRIPTION

The UNCATALOG\$OBJECT system call deletes an entry from the object directory of the specified job.

CONDITION CODES

- |                         |                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------|
| E\$OK                   | No exceptional conditions.                                                                                    |
| E\$CONTEXT              | The specified object directory does not contain an entry with the designated name.                            |
| E\$EXIST                | The job parameter is neither zero nor a token for an existing object.                                         |
| E\$NOT\$CON-<br>FIGURED | This system call is not part of the present configuration.                                                    |
| E\$PARAM                | The first byte of the STRING pointed to by the name parameter contains a value greater than 12 or equal to 0. |
| E\$TYPE                 | The job parameter is a token for an object that is not a job.                                                 |



## NUCLEUS SYSTEM CALLS

## WAIT INTERRUPT

WAIT\$INTERRUPT is used by an interrupt task to signal its readiness to service an interrupt.

|                                                |
|------------------------------------------------|
| CALL RQ\$WAIT\$INTERRUPT (level, except\$ptr); |
|------------------------------------------------|

## INPUT PARAMETER

level            A WORD containing an interrupt level which is encoded as follows (bit 15 is the high-order bit):

| <u>Bits</u> | <u>Value</u>              |
|-------------|---------------------------|
| 15-7        | 0                         |
| 6-4         | the interrupt level (0-7) |
| 3           | 1                         |
| 2-0         | 0                         |

## OUTPUT PARAMETER

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

## DESCRIPTION

The WAIT\$INTERRUPT system call is used by interrupt tasks immediately after initializing and immediately after servicing interrupts. Such a call places an interrupt task in the asleep state until reawakened by the interrupt handler for the same level. Each call that an interrupt task makes to WAIT\$INTERRUPT enables the given level.

## CONDITION CODES

E\$OK            No exceptional conditions.

E\$CONTEXT      The calling task is not the interrupt task for the given level.

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

E\$PARAM        The level parameter is invalid.



## CHAPTER 11. TERMINAL HANDLER

### GENERAL INFORMATION

The Terminal Handler supports real-time, asynchronous I/O between an operator's terminal and tasks running under the iRMX 86 Nucleus. It is intended for use in applications which require only limited I/O through a terminal, and it generally is used in applications that do not include the iRMX 86 I/O System. The features of the Terminal Handler include the following:

- Line editing capabilities.
- Keystroke control over output, including output suspension and resumption, and deletion of data being sent by tasks to the terminal.
- Echoing of characters as they are entered into the Terminal Handler's line buffer.

An output-only version of the Terminal Handler is available for use in applications in which tasks send output to a terminal but do not receive input from the terminal.

The remainder of this chapter is divided into two parts. The first part, Using a Terminal with the iRMX 86 Operating System, provides the information that is needed by an operator of the terminal. The second part, Programming Considerations, contains the information that a programmer needs to write tasks that send data to, or receive data from, the terminal. In the first part, there are a few references to the mailboxes that tasks use to communicate with the terminal. If you are puzzled by such a reference, look in the second part for an explanation.

### USING A TERMINAL WITH THE iRMX 86 OPERATING SYSTEM

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

## TERMINAL HANDLER

characters. In the remainder of this section, the handling of these two types is discussed, and the significance of each of the special characters is explained.

### HOW NORMAL CHARACTERS ARE HANDLED

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

### HOW SPECIAL CHARACTERS ARE HANDLED

Table 11-1 lists the special characters and summarizes the effects of each of them. The following text comprises complete descriptions of the effects of the special characters. In these descriptions, there are several references to "the current line." The current line is the contents of the MESSAGE CONTENT field of the input request message currently being processed.

Table 11-1. Special Character Summary

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

## TERMINAL HANDLER

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

### Rubbing Out a Previously-Typed Character (RUBOUT)

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

**External Effects:** If the current line is empty, the BEL character (07H) is sent to the terminal. Otherwise, the character is "rubbed out" in accordance with one of two available rubout modes. In the copying mode, the character being deleted from the current line is re-echoed to the display. For example, entering "CAT" and then striking RUBOUT three times results in the display "CATTAC". In the blanking mode, the deleted character is replaced on the CRT screen with the blanking character. For example, entering "CAT" and then striking RUBOUT three times deletes all three characters from the display. The copy mode is the default mode. The default blanking character for the blanking mode is a space (20H). If you wish to change either of these defaults, refer to the iRMX 86 Configuration Guide for ISIS-II Users (Manual order number 9803126) or talk to someone who knows about configuration.

### Displaying the Current Line (control-R)

**Internal Effects:** None.

**External Effects:** Sends a carriage return and line feed to the terminal, followed by the current line. If the current line is empty, the previous line is sent to the display, where it can be line edited and submitted as a new input message.

### Deleting the Current Line (control-X)

**Internal Effects:** Empties the current line.

**External Effects:** Causes the sequence (#, Carriage Return, Line Feed) to be sent to the terminal.

## TERMINAL HANDLER

### Sending an Empty Message (control-Z)

Internal Effects: Puts a zero in the ACTUAL field of the input request message currently being processed. The message is then sent to the appropriate response mailbox.

External Effects: None.

### Signalling the End of a Line of Input (Carriage Return, Line Feed, or ESCape)

Internal Effects: Puts either the ASCII end-of-transmission character (OAH in the case of Carriage Return or Line Feed) or the ESCape character (lBH) in the current line. Each of these characters signals the end of a message, so the input request message currently being constructed is sent to the appropriate response mailbox.

External Effects: If the end-of-line indicator is either Carriage Return or Line Feed, both Carriage Return and Line Feed are sent to the terminal. If the indicator is ESCape, however, there is no effect on the display.

## OUTPUT CONTROL

Output request messages that are sent to output mailbox RQTHNORMOUT can be processed in three ways:

- They can be output as described later under Programming Considerations.
- They can be queued at RQTHNORMOUT where they remain until an operator at the terminal takes action to permit processing of the messages.
- They can be discarded.

In the descriptions that follow, these methods of dealing with normal output requests are called the normal mode, the queueing mode, and the suppression mode, respectively. Initially, output is in the normal mode.

### Suspending Output (control-S)

Puts normal output in the queueing mode.

## TERMINAL HANDLER

### Resuming Output (control-Q)

Negates the effects of control-S by allowing output requests that are queued at RQTHNORMOUT to be displayed.

### Deleting or Restarting Output (control-O)

If output is in the normal mode, control-O puts it in the suppression mode. If output is in the suppression mode, control-O restores it to the normal mode. If output is in the queuing mode, control-O has no effect.

## PROGRAM CONTROL

### Calling a User-written Procedure Manually (control-C)

Control-C invokes a user-written procedure called RQ\$ABORT\$AP. This procedure can perform any actions that suit the application. However, control-C is normally used to abort an application. Control-C also causes the effects produced by control-Z; that is, it returns the current input request message with its ACTUAL field set to zero.

## SETTING A BAUD RATE

The Terminal Handler can be set to operate at any of the following baud rates:

110  
150  
300  
600  
1200  
2400  
4800  
9600  
19200

The rate is set during software configuration, when an iRMX 86 system variable, RQRATE, is assigned one of the previously mentioned baud rate values.

## TERMINAL HANDLER

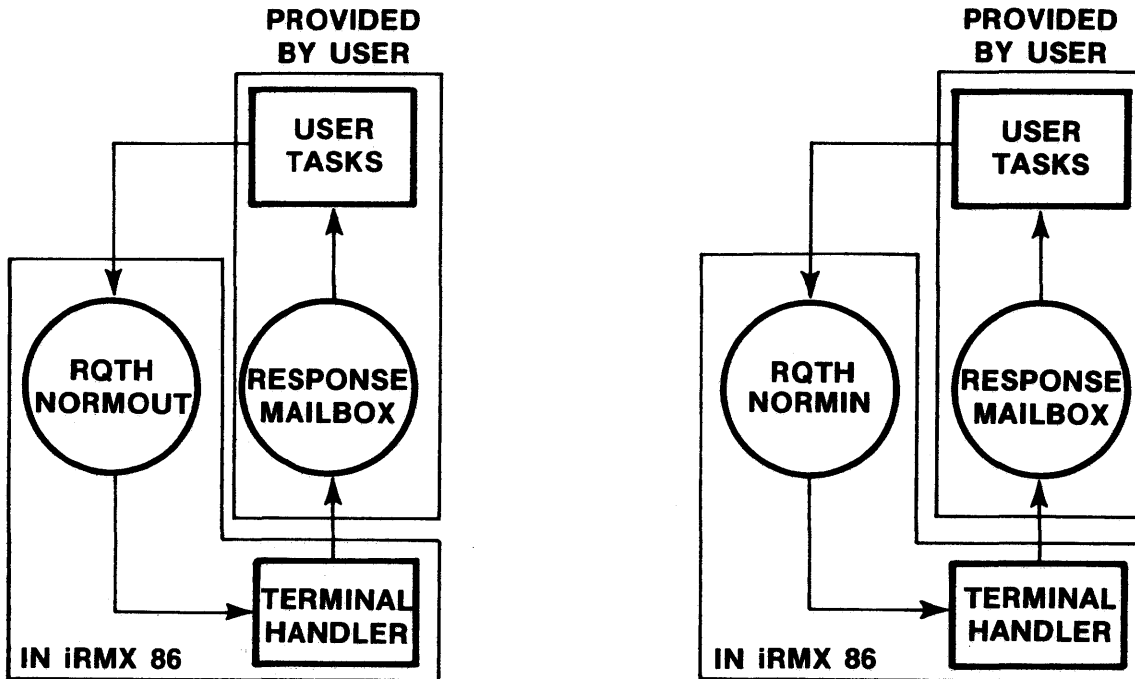


Figure 11-1. Input and Output Mailbox Interfaces.

### PROGRAMMING CONSIDERATIONS

The iRMX 86 Terminal Handler supports terminal input and output by providing mailbox interfaces. Figure 11-1 shows the mailboxes that are used typically. In the figure, an arrow pointing from a task to a mailbox represents a SEND\$MESSAGE system call. An arrow pointing from a mailbox to a task indicates a RECEIVE\$MESSAGE system call.

The protocol that tasks observe is much the same for input and output. In each case, the task initiates I/O by sending a request message to a mailbox. An input request mailbox, RQTHNORMIN, and an output request mailbox, RQTHNORMOUT, are provided. The Terminal Handler processes the request and then sends a response message back to the requesting task. The task waits at a response mailbox for the message. Thus, when a task does either input or output, it sends and then receives. The full details of the input and output protocols are described later in this section.

~~For both input and output, the medium is the message segment sent by a task to the Terminal Handler.~~ The format of a request message is depicted in Figure 11-2. The numbers in that figure are offsets, in bytes, from the beginning of the segment. The field names have different meanings for input and for output. For both input and output, the first four fields are WORD values. The MESSAGE CONTENT field can be up to 132 bytes in length for input and up to 65527 bytes in length for output.



## TERMINAL HANDLER

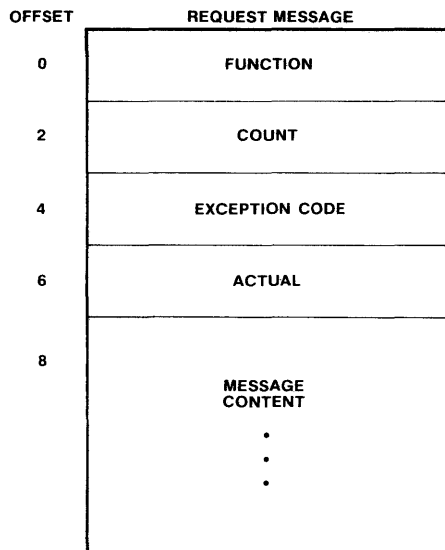


Figure 11-2. Request Message Format.

In the following discussions, the names ~~F\$WRITE~~ and ~~F\$READ~~ literal names for the particular WORD values 5 and 1, respectively.

### OUTPUT

The first thing a task does when transmitting output is prepare an output request message. The task must fill in the following fields prior to sending the message:

FUNCTION --- F\$WRITE. 5

COUNT --- the number of bytes (not to exceed 65527) in the MESSAGE CONTENT field.

MESSAGE CONTENT --- the bytes that are to be output.

Having prepared the message segment, the task must send it to the output request mailbox RQTHNORMOUT. Messages sent to this mailbox are processed in a first-in-first-out manner. Processing a message involves sending the characters in the MESSAGE CONTENT field to the terminal until a total of COUNT characters have been sent. There is one exception; when the Terminal Handler encounters the end-of-transmission character (OAH), it sends a Carriage Return and a Line Feed to the terminal.

When sending the output request message, the task specifies a user-supplied response mailbox. If no response mailbox is specified, the Terminal Handler will delete the segment that contained the message. In addition to transmitting the message

## TERMINAL HANDLER

to the terminal, the Terminal Handler fills in the remaining fields in the output request message. The requesting task can wait at the response mailbox (that is, it can call the RECEIVE\$MESSAGE system call with a time limit of OFFFFH) immediately after sending the output request. By observing this protocol, the task can learn of the success or failure of the output attempt. The fields that provide this information are the following:

- EXCEPTION CODE --- the encoded result of the output operation:
  - E\$OK --- the operation was successful.
  - E\$PARAM --- the FUNCTION field in the message did not contain F\$WRITE.
  - E\$BOUNDS --- the COUNT field in the message is too big for the segment, that is, COUNT + 8 is greater than the length of the segment containing the message.
- ACTUAL --- the actual number of bytes output.

In summary, the protocol observed by tasks doing output is as follows:

- Prepare the output request message segment, filling in the FUNCTION, COUNT, and MESSAGE CONTENT fields.
- Send the segment, via the SEND\$MESSAGE system call, to RQTHNORMOUT. It is advisable, but not necessary, to specify a response mailbox in the system call.
- Wait indefinitely, via the RECEIVE\$MESSAGE system call, at the response mailbox. When received, the message contains the results of the transmission in the EXCEPTION CODE and ACTUAL fields.

## INPUT

The protocol for obtaining input is much the same as that for outputting. A message is prepared and sent to a request mailbox, then the message is received at a response mailbox. There is a significant difference, however. The input is contained in the message segment at the response mailbox, so designating a response mailbox and then waiting there is required.

## CAUTION

When multiple tasks use the same mailbox for input from the terminal, there is a chance that a task will get input that is intended for another task.

## TERMINAL HANDLER

The first thing a task needing input does is prepare an input request message. ~~It must fill in the FUNCTION and COUNT fields prior to sending its request.~~ The FUNCTION field must contain F\$READ. The COUNT field reflects the maximum possible number of input characters in the input message. ~~The value of COUNT must not exceed 132; moreover, COUNT + 8 must not exceed the length of the input request message segment.~~

When sending the input request message, the task ~~must specify the response mailbox in its call to the SEND\$MESSAGE system call.~~ The Terminal Handler obtains characters from the terminal and places them in the MESSAGE CONTENT field. The message is ~~terminated by an end of line character (Carriage Return, Line Feed, or ESCape).~~ The lone exception is when the end-of-line character has been "normalized" by being preceded by a control-P; then the end-of-line character is treated as a normal character.

### NOTE

If more than COUNT characters are entered prior to the end-of-line character, the extra characters are ignored, and the terminal beeps at the operator.

After the message is complete, the Terminal Handler fills in the EXCEPTION CODE and ACTUAL fields as follows:

- EXCEPTION CODE --- the encoded result of the input operation, which is one of the following:
  - E\$OK --- the operation was successful.
  - E\$PARAM --- either the FUNCTION field in the message did not contain F\$READ or the COUNT field was greater than 132.
  - E\$BOUNDS --- COUNT + 8 is greater than the length of the message segment.
- ACTUAL --- the number of bytes actually entered and placed in the MESSAGE CONTENT field.

The requesting task must wait (that is, it must make a RECEIVE\$OBJECT system call with a time limit of OFFFFH) at the designated response mailbox immediately after sending the input request.

In summary, the input protocol is as follows:

- Prepare the input request message segment, filling in the FUNCTION and COUNT fields.

## TERMINAL HANDLER

- Send the segment, via the SEND\$MESSAGE system call, to RQTHNORM\$IN. In the call, specify a response mailbox.
- Wait indefinitely, via the RECEIVE\$MESSAGE system call, at the response mailbox. When received, the message segment will contain the results of the input operation in the MESSAGE CONTENT, EXCEPTION CODE, and ACTUAL fields.

## CHAPTER 12. DEBUGGER

### GENERAL INFORMATION

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 monitor in the iSBC 86/12A Single Board Computer.

The Debugger supplies its own Terminal Handler, which includes all of the capabilities described in Chapter 11. If your application includes the Debugger, then you must use its Terminal Handler, rather than linking in the Terminal Handler module.

### DEBUGGER CAPABILITIES

The iRMX 86 Debugger enables you to do the following:

- View iRMX 86 system lists, including the lists of the jobs, the tasks, the ready tasks, the suspended tasks, the task queues at exchanges, the object queues at mailboxes, the exchanges, and the iRMX 86 segments.
- Inspect jobs, tasks, exchanges, and segments.
- 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.
- View the list of special debugging variables.
- Deactivate the Debugger.

## 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, manual order number 9800714.

### DEBUGGING CAPABILITIES IN THE iSBC 86/12A MONITOR

The iSBC 86/12A monitor has 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.

### INVOKING THE DEBUGGER

The Debugger is invoked when you enter control-D at the terminal. The Debugger responds with its sign-on message, "iRMX 86 DEBUGGER V2.0", and its prompt character, an asterisk.

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 later in this chapter.

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 gets

## DEBUGGER

the Debugger as its exception handler either by using the SET\$EXCEPTION\$HANDLER system call or when the task is created by means of either CREATE\$TASK or CREATE\$JOB. An example of code setting up one of these calls is the following:

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

EXCEPT$BLOCK.EXCEPT$PROC = @RQ$DEBUGGER$EX;
EXCEPT$BLOCK.EXCEPT$MODE = ZEROONETWOORTHREE;
.
.
.

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

### DEBUGGER INPUT AND OUTPUT

The Debugger obtains input one line at a time from its Terminal Handler. The end-of-line indicators are Carriage Return, Line Feed, and ESCape. When either Carriage Return or Line Feed is entered, the current input line is sent to the Debugger; when ESCape is entered, the current input line is discarded and a prompt is displayed.

The Debugger generates display at the terminal by sending output messages to its Terminal Handler. To suppress output from application tasks during a debugging session, type control-S. If control-S is not entered, any output from tasks is interspersed with output from the Debugger. To allow resumption of output from tasks, type control-Q. Control-S and control-Q have no effect on output from the Debugger.

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 diverts all output including Debugger prompts from the terminal to the bit bucket until you enter another control-O.

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 OFFFH, 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.

## DEBUGGER

### SYNTAX OF DEBUGGER COMMANDS

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

Debugger commands fall into families. The syntax for each family is described in detail later in this chapter.

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, 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 of the PL/M-86 variety.

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

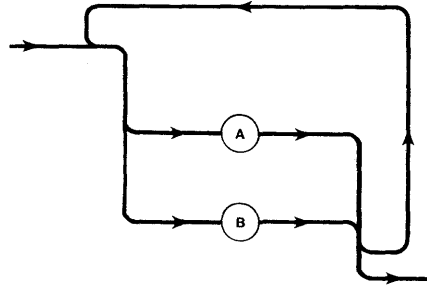
- immediately after a command key that is not a name and
- between a letter or digit and a non-letter, non-digit. The legal characters of the latter type are the following: ; @ = / \ : ( ) \* + - ,

### PICTORIAL REPRESENTATION OF SYNTAX

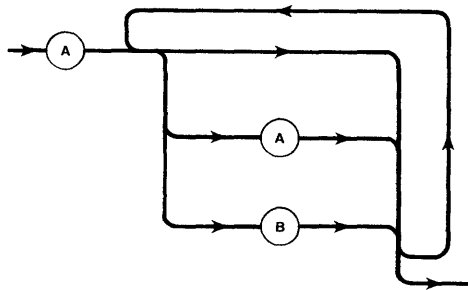
In this chapter, a schematic device illustrates the syntax of commands. The schematic consists of what looks like an aerial view of a model railroad setup, with syntactic entities scattered along the track. Imagine that a train enters the system at the upper left, drives around as much as it can or wants 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



## DEBUGGER



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 chapter, 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 OAH is a constant.

## DEBUGGER

### NOTE

If more than four hexadecimal digits compose a constant, only the low order four digits are used. Binary, octal, and decimal constants are not understood by the Debugger.

- **NAME.** A name is a period followed by up to 31 alphabetic or numeric characters, the first of which must be alphabetic.

Examples:

.task

.mailbox7

- **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 12-1:

### 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.

# DEBUGGER

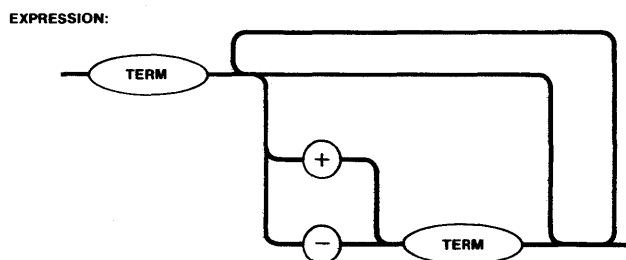
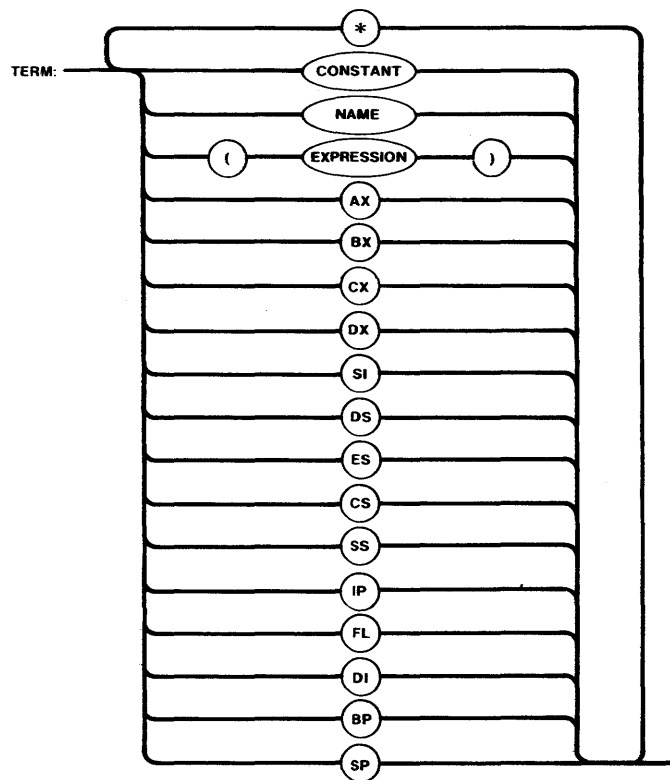


Figure 12-1. Syntax Diagrams for Term and Expression

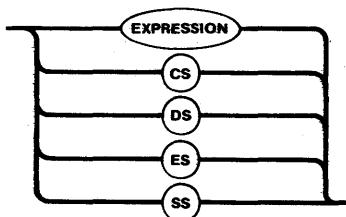


Figure 12-2. Syntax Diagram for Item

## DEBUGGER

- 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 12-2.

### DEBUGGER COMMANDS

This section presents the details of the Debugger commands. The commands fall into two groups, those relating to breakpoints and those that enable you to observe or change the contents of memory.

### BREAKPOINT CONTROL

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. 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". Moreover, depending on the breakpoint, the tasks in the containing job might be 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.

## DEBUGGER

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 by the BT command (described later), rather than by incurring a breakpoint. Another way in which a task can enter the N (for null) state is when another task in the same job incurs a breakpoint which had been set with the DB command (also 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.

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

- The display format for an execution breakpoint is

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 segment in which the breakpoint was set.                      |
| iiii   | The offset of the breakpoint within its segment.                              |

- The display format for an exchange breakpoint is

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

## DEBUGGER

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 or task whose token follows.                                                                                                                                                                                                                                                                                                                                                                                                                           |
| xxxx   | A token for the exchange.                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| e      | Indicates the type of the exchange (M for mailbox, S for semaphore).                                                                                                                                                                                                                                                                                                                                                                                                                               |
| tttt   | A token for the task.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| item   | One of the following:<br><br>A pair of tokens, jjjjJ/oooo, with oooo being a token for the object being sent or received, t indicating the type of the object (J for job, T for task, M for mailbox, S for semaphore, and G for segment), and jjjj being a token for the object's containing job, if the exchange is a mailbox. If the kind of operation was receive, but no object was there to be received, item is 0000.<br><br>The number of units held by the exchange, if it is a semaphore. |

- The display format for an exception breakpoint is

EXCEPTION: jjjjJ/ttttT, CS=cccc, IP=iiii, TYPE=www, PARAM=vvv

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. |

## DEBUGGER

|     |                                                                                                                                                                               |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| www | The numerical value of the exception code; reflects the nature of the exceptional condition. Chapter 8 contains the mnemonic condition codes and their numerical equivalents. |
| vvv | The number (0001 for first, 0002 for second, etc.) of the parameter that caused the exceptional condition. If no parameter was at fault, vvv is 0000.                         |

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 delete the breakpoint by declaring a different exception handler.
- 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.
- Exception breakpoints are set for a "kind" of event, namely the occurrence of an exceptional condition when the task that set the breakpoint makes a system call. An execution or exchange breakpoint, on the other hand, is set at a "place."
- An exception breakpoint is not known to the Debugger by a breakpoint variable name.

If you want to monitor a particular task, you can designate it to be the breakpoint task. If the task is not already on the breakpoint list when you do this, it is suspended and is placed on the breakpoint list with a null breakpoint state. After designating a breakpoint task, you can examine some of its registers but not alter them. You can also ascertain the breakpoint state of the task. When ready, you can easily resume the task and remove it from the breakpoint list.

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.

### Establishing a Breakpoint --- The DB Command

#### Syntax

The syntax for the DB command is given in figure 12-3.

## DEBUGGER

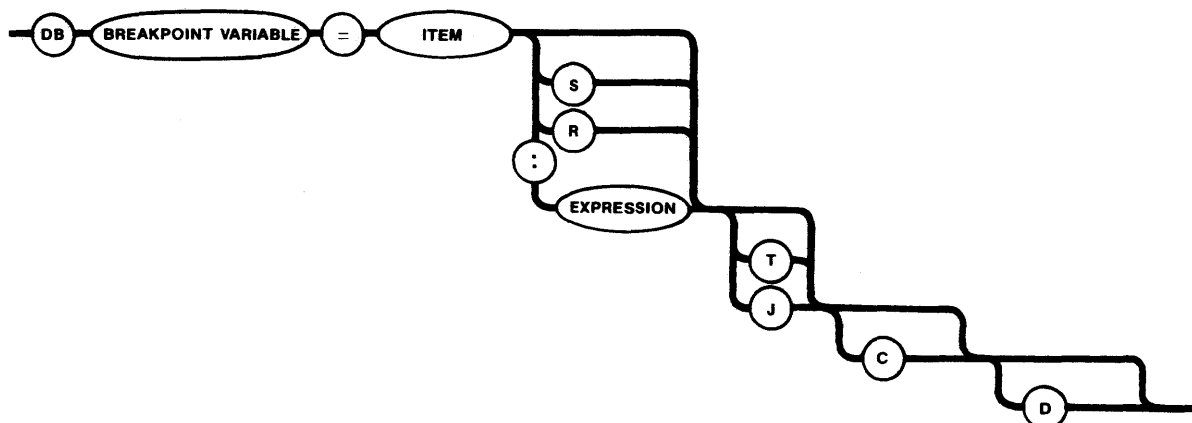


Figure 12-3. Syntax Diagram for Establishing a Breakpoint

### Explanation of Syntax

**BREAKPOINT VARIABLE** A Debugger name. If the Debugger's symbol table already contains this name, an error message will appear on the terminal's display.

**ITEM** If followed by ":" and an **EXPRESSION**, you are setting an execution breakpoint, and **ITEM** must contain a token for a segment, while **EXPRESSION** must contain an offset. Otherwise, you are setting an exchange breakpoint, and **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**.

**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**.



## DEBUGGER

- C Directs the Debugger not to "break" tasks that incur the breakpoint, and not to put them on the breakpoint list.
- D Directs the Debugger to delete the breakpoint when it is first incurred by a task. The D option does not suppress the display that results when a task incurs a breakpoint.

### Effects

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 to alter or delete the breakpoint.

### Changing a Breakpoint

#### Syntax

The syntax for this command is given in Figure 12-4.

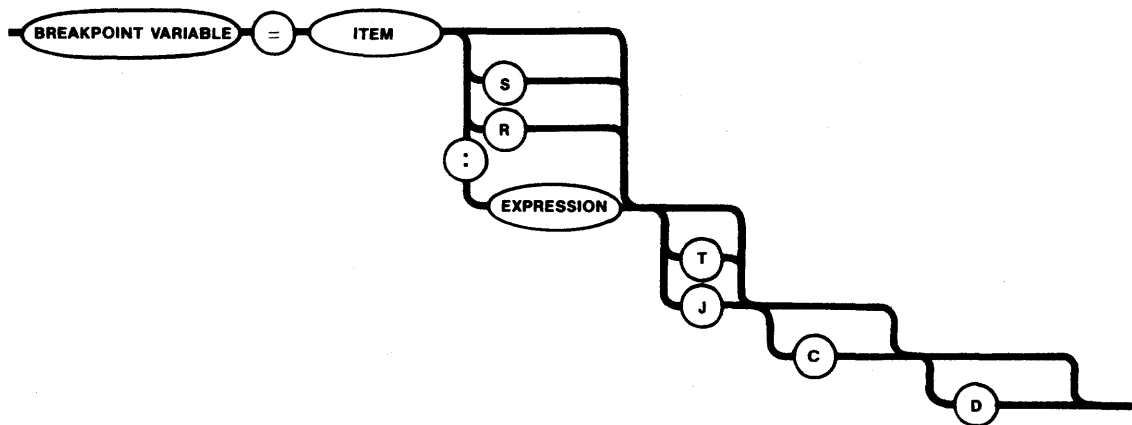


Figure 12-4. Syntax Diagram for Changing a Breakpoint

#### Explanation of Syntax

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

## DEBUGGER

- ITEM If followed by ":" and an EXPRESSION, you are setting an execution breakpoint, and ITEM must contain a token for a segment, while EXPRESSION must contain an offset. Otherwise, you are setting an exchange breakpoint, and 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.
- 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 Directs the Debugger not to "break" tasks that incur the breakpoint, and not to put them on the breakpoint list.
- D Directs the Debugger to delete the breakpoint when it is first incurred by a task. The D option does not suppress the display that results when a task incurs a breakpoint.

### Effects

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 to delete or change the breakpoint.

### Deleting a Breakpoint --- The Z Command

#### Syntax

The syntax for the Z command is given in Figure 12-5.



Figure 12-5. Syntax Diagram for Deleting a Breakpoint

## DEBUGGER

### Explanation of Syntax

**BREAKPOINT VARIABLE** A Debugger name. If the Debugger's symbol table does not contain the name, or if it does contain the name but the name is not stored as a breakpoint variable, an error message is displayed.

### Effects

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

### Examining a Breakpoint

#### Syntax

The syntax for this command is given in Figure 12-6.



Figure 12-6. Syntax Diagram for Examining a Breakpoint  
Explanation of Syntax

**BREAKPOINT VARIABLE** The name of the breakpoint to be examined.

### Effects

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

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

#### where

|        |                                                                                                                                                                                                    |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| bp-var | The name of the breakpoint variable.                                                                                                                                                               |
| xxxx   | The base of the segment containing the instruction at which the breakpoint is set.                                                                                                                 |
| yyyy   | The offset of the instruction within the segment.                                                                                                                                                  |
| z      | Indicates whether a 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. |

## DEBUGGER

ops                                    If any are present, can be C (for Continue task) and/or D (for Delete breakpoint).

If the breakpoint is an exchange breakpoint, the following display is sent 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 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    | If any are present, can be C (for continue task) and/or D (for delete breakpoint).                                                                                                                 |

You cannot examine an exception breakpoint.

### Viewing the Breakpoint List --- The BL Command

#### Syntax

The syntax for the BL command is given in Figure 12-7.



Figure 12-7. Syntax Diagram for Viewing the Breakpoint List.

#### Effects

The Debugger responds to this command by displaying the entire breakpoint list as follows:

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

## DEBUGGER

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). |

### Viewing the Breakpoint Parameters --- The B Command

Syntax

The syntax for the B command is given in Figure 12-8.



Figure 12-8. Syntax Diagram for Viewing the Breakpoint Parameters

Effects

The B command has three separate effects. The first is to display the breakpoint list as

```
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). |

The second effect of the B command is to display the breakpoint task as

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

where

|      |                                                     |
|------|-----------------------------------------------------|
| jjjj | A token for the job containing the breakpoint task. |
|------|-----------------------------------------------------|

## DEBUGGER

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

The third and final effect of the B command is to display the breakpoint variables intermixed in either of two formats. Execution breakpoints are displayed as

bp-var = xxxx:yyyy z ops

where:

bp-var        The name of the breakpoint variable.

xxxx         The base of the segment containing the instruction at which the breakpoint is set.

yyyy         The offset of the instruction within the segment.

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          If any are present, 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          If any are present, can be C (for Continue task) and/or D (for Delete breakpoint).

Removing a Task from the Breakpoint List --- The G Command

Syntax

The syntax for the G command is given in Figure 12-9.

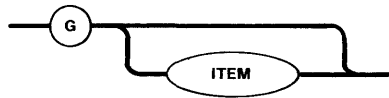


Figure 12-9. Syntax Diagram for Removing a Task from the Breakpoint List

Explanation of Syntax

|      |                                                                                                                                             |
|------|---------------------------------------------------------------------------------------------------------------------------------------------|
| ITEM | A token for a task on the breakpoint list. If the given token is not for a task on the breakpoint list, an error message will be displayed. |
|------|---------------------------------------------------------------------------------------------------------------------------------------------|

Effects

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 removes the designated task from the breakpoint list. 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 --- The BT Command

Syntax

The syntax for this command is given in Figure 12-10.

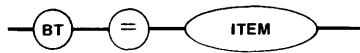


Figure 12-10. Syntax Diagram for Establishing the Breakpoint Task

Explanation of Syntax

ITEM                    A token for an existing task.

Effects

The task designated by ITEM becomes the breakpoint task. If it is not already on the breakpoint list, it is suspended and placed on the breakpoint list.

Inquiring as to the Breakpoint Task --- The BT Command

Syntax

The syntax for this command is given in Figure 12-10.



Figure 12-11. Syntax Diagram for Inquiring as to the Breakpoint Task

This command causes the following to be displayed at the terminal:

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).



## DEBUGGER

If there is no breakpoint task, the display is

BT=0

### Viewing the Breakpoint Task's Registers --- The R Command

#### Syntax

The syntax for this command is given in Figure 12-12.

#### Explanation of Syntax

|     |                                                                                                                                  |
|-----|----------------------------------------------------------------------------------------------------------------------------------|
| R   | The command key letter. By itself, it represents a request for the display of all the breakpoint task's iAPX 86 register values. |
| Ryy | Represents a request for the display of only the breakpoint task's yy register value.                                            |

#### Effects

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, then the contents of the breakpoint task's 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.

DEBUGGER

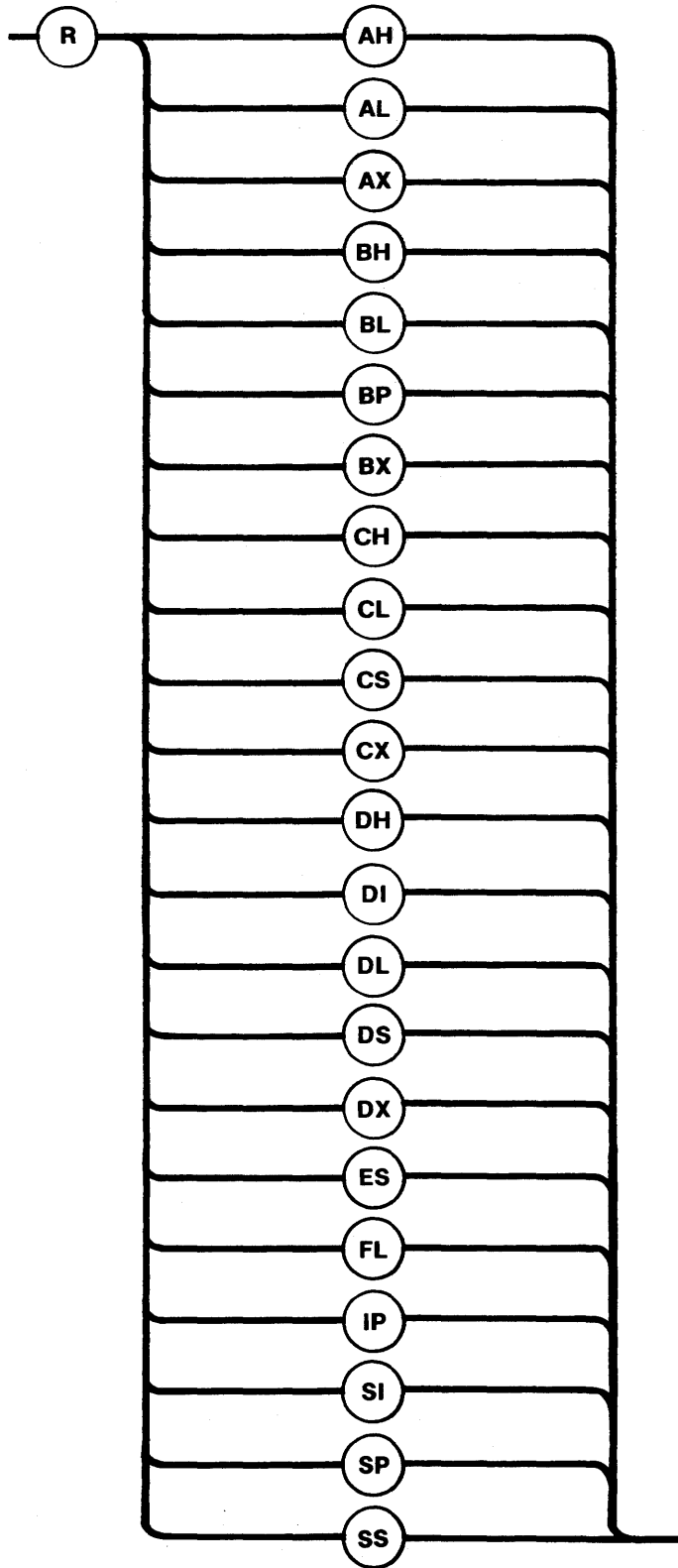


Figure 12-12. Syntax Diagram for Viewing the Breakpoint Task's Registers

Altering the Breakpoint Task's Registers --- The R Command

Syntax

The syntax for this command is given in Figure 12-13.

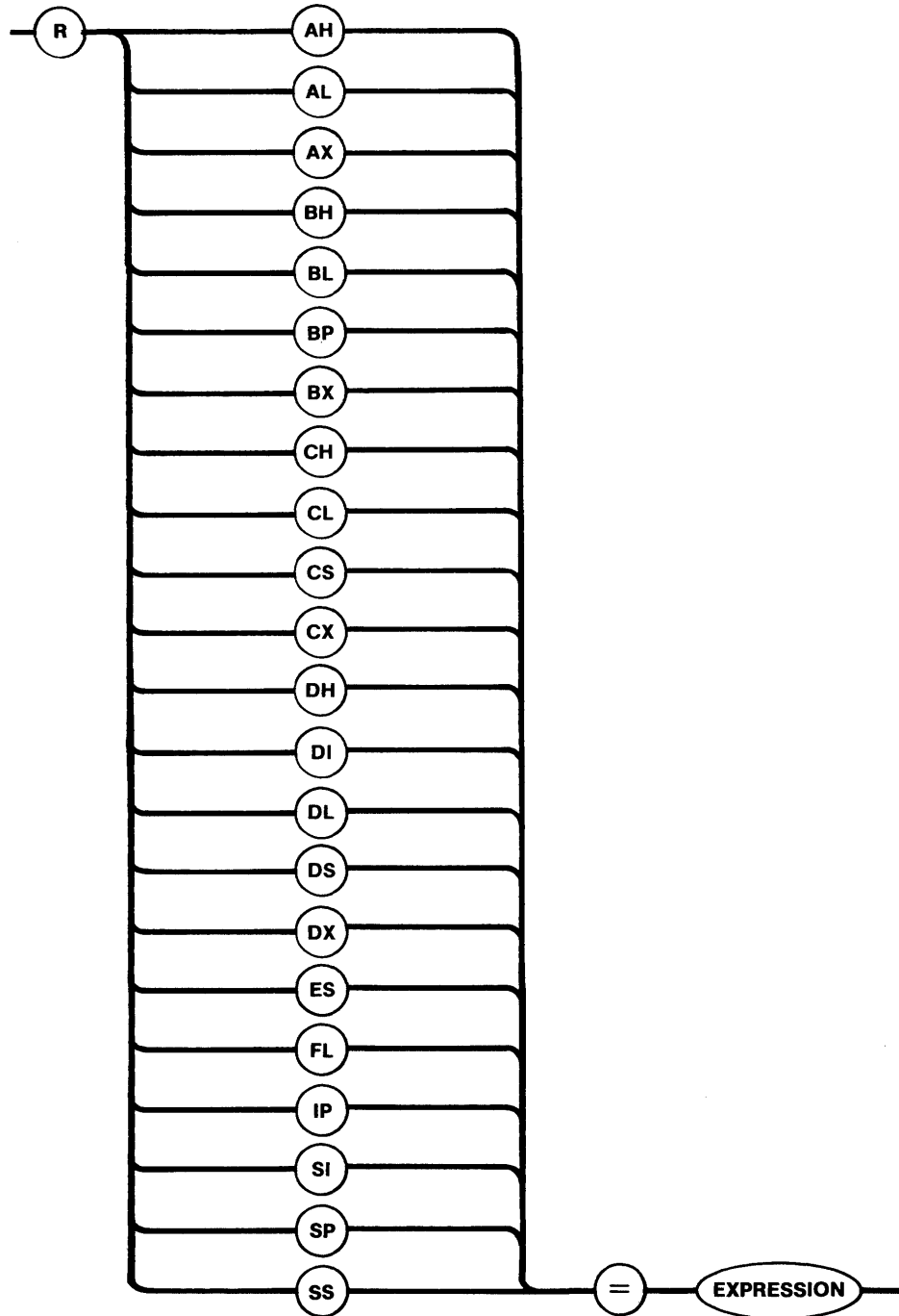


Figure 12-13. Syntax Diagram for Altering the Breakpoint Task's Registers

## DEBUGGER

### Explanation of Syntax

Ryy Signals a request that the contents of the breakpoint task's yy register are to be changed.

### Effects

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.

### OBSERVATION AND MANIPULATION COMMANDS

The commands in this section enable you to inspect or modify the contents of absolute memory locations, to view the iRMX 86 system lists, and to inspect tasks, jobs, exchanges, and segments.

In the descriptions of the commands for examining and modifying memory locations, 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 not represent a printable ASCII character, and the current display mode is A, then the Debugger prints a period. The initial current display mode is B.

## DEBUGGER

- The value of the current segment base is always the value of the most recently used 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.
- 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. In case 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).

### Examining or Modifying Memory --- The M Command

#### Syntax

The syntax for the M command is given in Figure 12-14.

#### Explanation of the M Command Options

The options in the M command fall into three categories:

- If an option begins with "!", then it is a request to redefine the current display mode.

## DEBUGGER

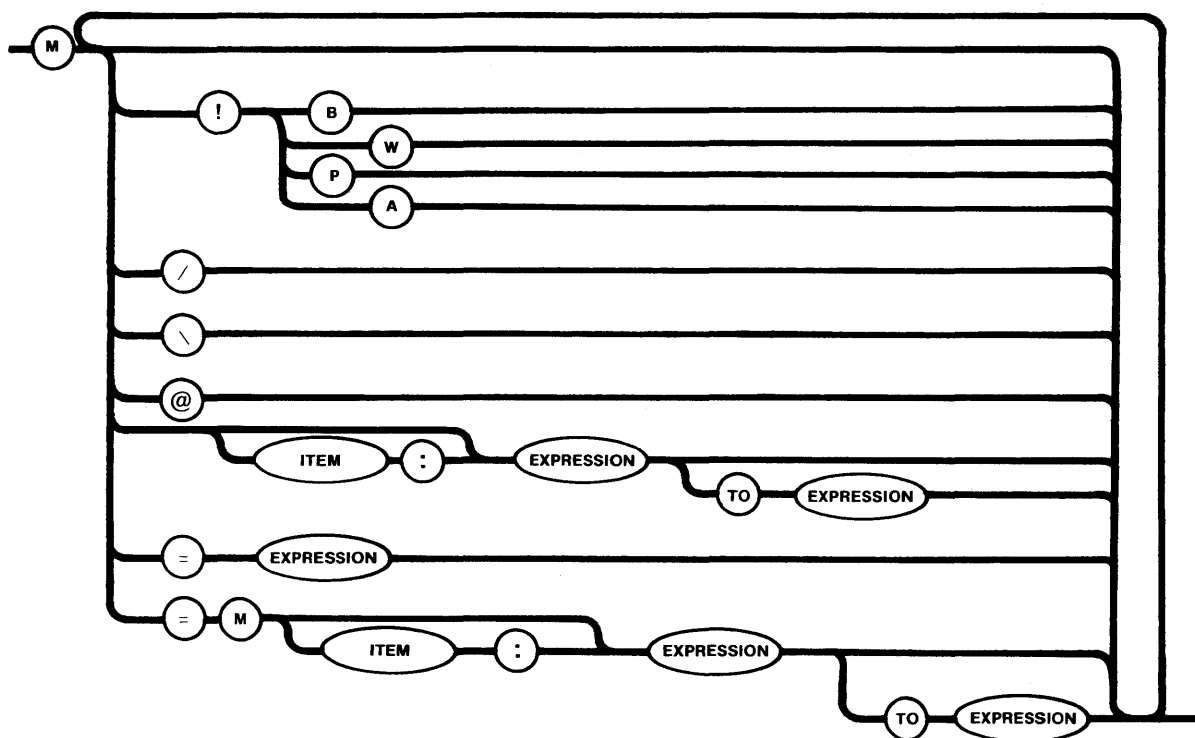


Figure 12-14. Syntax Diagram for Examining or Modifying Memory

- If an option begins with "=", it is a request to alter memory
- The remaining options are requests for display of memory.

In what follows, the options of the M command are described with examples. As much as possible, effects are described separately for each option. When combinations of options produce special effects, these effects are described.

As the syntax diagram reveals, it is permissible to make multiple requests in the same command line. The Debugger treats multiple requests as if they had been submitted one at a time. For example, M/// is treated as three M/ requests. In the following descriptions, it is assumed that each command line consists of only one request.

You must separate the elements in a command with a space. An exception is that ITEM, colon, and EXPRESSION may be contiguous, like 4C67:F325.

### NOTE

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

## DEBUGGER

- 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 modify a location that is in ROM, an "attempt to modify non-RAM location" message appears on the display. If you attempt to read or write to a non-existent 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.

Each description is followed by an example in which the request and resulting display are shown exactly as they would appear at the terminal. Assume, for each example request, that the following circumstances exist when the request is made:

- The following memory locations contain the indicated values:

```
base:0300 21 47 E2 C8 31
 offset: 2643 2644 2645 2646 2647
```

```
base:0400 01 02 03 0F 10 ... 16
 offset: 0000 0001 0002 000E 000F 0A09
```

- The current address is 0400:0008, that is, the current segment base is 0400 and the current offset is 0008.
- The current display mode is byte.

## DEBUGGER

### Options for Setting the Current Display Mode

M!B --- This option sets the current display mode to byte.

M!W --- This option sets the current display mode to word.

M!P --- This option sets the current display mode to pointer.

M!A --- This option sets the current display mode to ASCII.

None of these requests results in an immediate display.

### Options for Displaying Memory

The options in this section all enable you to ascertain the contents of memory without disturbing those contents. Be aware, however, that all of these options change the current offset, and some of them change the current segment base. None changes the current display mode.

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

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

Example: M\  
0400:0007 08

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

M@ --- This option sets the current offset equal to the value at the current address. Then the value at the adjusted current address is displayed.

Example: M@  
0400:0A09 16



## DEBUGGER

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

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

Example: M 300:2644  
0300:2644 47

M EXPRESSION TO EXPRESSION --- This option displays the values in a series of consecutive locations. The expressions determine the beginning and ending offsets, respectively. The current segment base is used as a base. After the display is output, the current offset is set 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  
0400:0004 05 06 07  
(2) M!W  
M 4 TO 6  
0400:0004 0605 0807

### Options for Altering Memory

The options in this section enable you to change 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.

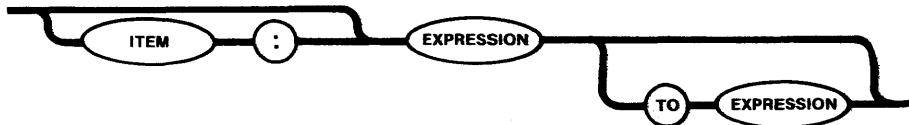
Unlike the displaying options of the previous section, the modifying options do not affect either the current segment base or the current offset.

## DEBUGGER

When executing the options of this section, 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.

The options of this section copy data in the byte mode. The current display mode is not affected by these copying options.

Some M command lines have the form `M<destination>=M<source>`, where both `<source>` and `<destination>` have the following syntax:



In the descriptions that follow, occasional references are made to options of this form because, when used, they can affect the results of the next invocation of an option for altering memory.

For convenience, we use the phrase "the previous option has a destination field" as an abbreviation for

"The following conditions are both true:

- The previous option used was of the `M<destination>=M<source>` variety.
- `<destination>` specifies a range of at least two addresses."

`M=EXPRESSION` --- This option can be used only if the current display mode is byte or word. `M=EXPRESSION` copies the value represented by `EXPRESSION` into the byte or word at the current address. However, if the previous option had a destination field, this option instead copies the value of `EXPRESSION` into each byte or word in destination .

Examples:

- (1) When the previous option did not have a destination field:

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

- (2) When the previous option had a destination field:

```
M 1 TO 4 = M 5 TO 8
```

## DEBUGGER

```
0400:0001 02 03 04 05
0400:0001 06 07 08 09
M = 4C
0400:0001 06 07 08 09
0400:0001 4C 4C 4C 4C
```

M=M EXPRESSION --- This option uses the current segment base and the offset indicated by the value of EXPRESSION to compute an address. It copies the value at that computed address into the location specified by the current address. However, if the previous option had a destination field, the value at the computed address is instead copied to the locations in the destination field.

### Examples:

- (1) When the previous option did not have a destination field:

```
M = M 4
0400:0008 09
0400:0008 05
```

- (2) When the previous option had a destination field:

```
M 1 TO 3 = M 5 TO 7
0400:0001 02 03 04
0400:0001 06 07 08
M = M 4
0400:0001 06 07 08
0400:0001 05 05 05
```

M=M ITEM:EXPRESSION --- This option uses ITEM and EXPRESSION as base and offset, respectively, to compute an address. M=M ITEM:EXPRESSION copies the value at that computed address into the location specified by the current address. However, if the previous option had a destination field, the value at the computed address is instead copied to the locations in the destination field.

### Examples:

- (1) When the previous option did not have a destination field:

```
M = M 300:2643
0400:0008 09
0400:0008 21
```

## DEBUGGER

- (2) When the previous option had a destination field:
- ```
M 1 TO 4 = M 5 TO 8
0400:0001 02 03 04 05
0400:0001 06 07 08 09
M = M 300:2643
0400:0001 06 07 08 09
0400:0001 21 21 21 21
```

M=M EXPRESSION TO EXPRESSION --- This option uses the current segment base and, in order, the offsets indicated by the EXPRESSIONs, to compute a beginning address and an ending address. M=M EXPRESSION TO EXPRESSION copies the sequence of values bounded by the computed addresses to the sequence of locations that begin at the current address. However, if the previous option had a destination field, the sequence of values bounded by the computed addresses is copied to the destination field, with the source values being truncated or repeated as required.

Examples:

- (1) When the previous option did not have a destination field:
- ```
M = M A TO C
0400:0008 09 0A 0B
0400:0008 0B 0C 0D
```
- (2) When the previous option had a destination field:
- ```
M 1 TO 4 = M 5 TO 8
0400:0001 02 03 04 05
0400:0001 06 07 08 09
M = M A TO C
0400:0001 06 07 08 09
0400:0001 0B 0C 0D 0B      (first value
                             repeated)
```

M=M ITEM:EXPRESSION TO EXPRESSION --- This 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 previous option had a destination field, 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.

DEBUGGER

- (1) When the previous option did not have a destination field:

```
M = M 300:2643 TO 2647
0400:0008 09 0A 0B 0C 0D
0400:0008 21 47 E2 C8 31
```

- (2) When the previous option has a destination field:

```
M 1 TO 4 = M 5 to 8
0400:0001 02 03 04 05
0400:0001 06 07 08 09
M = M 300:2643 TO 2647
0400:0001 06 07 08 09
0400:0001 21 47 E2 C8      (last value
                             truncated)
```

Examining System Objects --- The I Command

Syntax

The syntax for the I command is given in Figure 12-15.

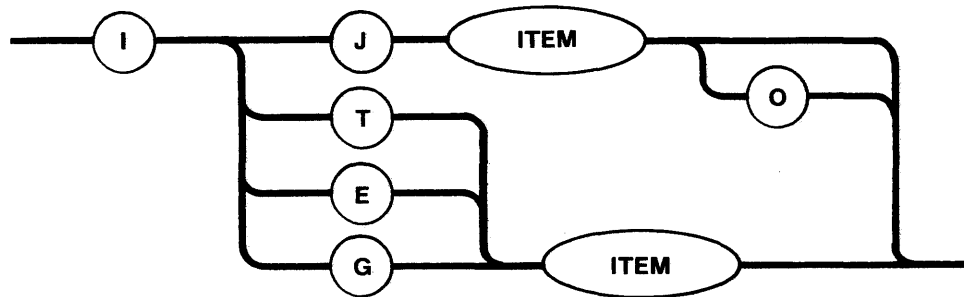


Figure 12-15. Syntax Diagram for Examining System Objects

Explanation of I Command Options

- J --- This option lists the principal attributes of the job whose token is represented by ITEM. If the O option is included, the object directory for the job is listed as well. In case there is a large number of entries in the object directory, the display might roll off the screen. To deal with this, use control-O, which works as described in Chapter 11. The form of the display is depicted in Figure 12-16.

DEBUGGER

```

----- 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 # CHILDREN JOBS gggg      DELETION PENDING    ppp
EXCEPTION MODE     hhhh          EXCEPTION HANDLER   qqqq:rrrr
MAXIMUM PRIORITY   iiii

NAME(S)  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa
         aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa

----- OBJECT DIRECTORY -----
MAXIMUM SIZE      uuuu          VALID ENTRIES      vvvv
NAME              TOKEN      NAME              TOKEN      NAME              TOKEN
ssssssssssss tttt      sssssssssssss tttt      sssssssssssss tttt

```

Figure 12-16. An iRMX 86™ Job Report

The following describes the fields in Figure 12-16:

Field	Meaning
aaaaaaaaaaaa	Each such field can contain 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	This is a hexadecimal token for the job.
cccc	This is the maximum number, in hexadecimal, of 16-byte paragraphs that the job's pool can contain.
dddd	This is the number of paragraphs that have been either allocated to tasks in the job or lent to child jobs.
eeee	This is the number, in hexadecimal, of existing objects in job bbbb.
ffff	This is the maximum number, in hexadecimal, of objects that can exist simultaneously in job bbbb.
gggg	This is the number, in hexadecimal, of existing jobs that are offspring of job bbbb.
hhhh	This is the exception mode for the job's default exception handler. Possible values are as follows:

DEBUGGER

<u>Value</u>	<u>When to Pass Control To Exception Handler</u>
0	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions
iiii	This is a hexadecimal value that indicates the maximum (numerically lowest) allowable priority for tasks in the job.
jjjj	This is a hexadecimal token for the parent of job bbbb. If job bbbb is the root job, however, jjjj is "ROOT".
kkkk	This is the minimum number, in hexadecimal, of 16-byte paragraphs that the job's pool can contain.
llll	This is the number, in hexadecimal, of unused 16-byte paragraphs in the job's initial pool.
mmmm	This is the number, in hexadecimal, of tasks currently in the job.
nnnn	This is the maximum number, in hexadecimal, of tasks that can exist simultaneously in job bbbb.
ppp	This 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	This is the base, in hexadecimal, of the start address of the job's default exception handler.
rrrr	This is the hexadecimal offset, relative to qqqq, of the start address of the job's default exception handler.
ssssssssss	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	This is the maximum allowable number, in hexadecimal, of entries in the job's object directory.

DEBUGGER

Field	Meaning
vvvv	This is the number, in hexadecimal, of entries currently in the job's object directory.

T --- This option displays the principal attributes of the task whose token is represented by ITEM. The form of the display is depicted in Figure 12-17.

```

----- iRMX86 TASK REPORT -----
TASK TOKEN          bbbb          CONTAINING JOB          jjjj
STACK SEGMENT BASE  cccc          STACK SEGMENT OFFSET   kkkk
STACK SEGMENT SIZE  dddd          STACK SEGMENT LEFT     llll
CODE SEGMENT BASE   eeee          DATA SEGMENT BASE     mmmm
INSTRUCTION POINTER ffff          TASK STATE             nnnnnnnn
STATIC PRIORITY     gggg          DYNAMIC PRIORITY       pppp
SUSPENSION DEPTH    hhhh          SLEEP UNITS REQUESTED  qqqq
EXCEPTION MODE      iiii          EXCEPTION HANDLER      rrrr:ssss

NAME(S)  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa
         aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa
    
```

Figure 12-17. An iRMX 86™ Task Report

The following describes the fields in Figure 12-17:

Field	Meaning
aaaaaaaaaaaa	Each such field can contain 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 printed here.
bbbb	This is a hexadecimal token for the task.
cccc	This is the base address, in hexadecimal, of the task's stack segment.
dddd	This is the size, in bytes, of the task's stack segment.
eeee	This is the base address, in hexadecimal, of the task's code segment.
ffff	This is the current value, in hexadecimal, of the task's instruction pointer.
gggg	This is the hexadecimal priority, as described in chapter 4, of the task.

DEBUGGER

Field	Meaning										
hhhh	This is the 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	This is the exception mode for the task's exception handler. Possible values are as follows:										
	<table border="0" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;"><u>Value</u></th> <th style="text-align: left;"><u>When to Pass Control To Exception Handler</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Never</td> </tr> <tr> <td>1</td> <td>On programmer errors only</td> </tr> <tr> <td>2</td> <td>On environmental conditions only</td> </tr> <tr> <td>3</td> <td>On all exceptional conditions</td> </tr> </tbody> </table>	<u>Value</u>	<u>When to Pass Control To Exception Handler</u>	0	Never	1	On programmer errors only	2	On environmental conditions only	3	On all exceptional conditions
<u>Value</u>	<u>When to Pass Control To Exception Handler</u>										
0	Never										
1	On programmer errors only										
2	On environmental conditions only										
3	On all exceptional conditions										
jjjj	This is a hexadecimal token for the task's containing job.										
kkkk	This is the hexadecimal offset, relative to cccc, of the task's stack segment.										
llll	This is the hexadecimal number of bytes currently available in the task's stack.										
mmmm	This is the base address, in hexadecimal, of the task's data segment.										
nnnnnnnn	This is the task's current execution state. Possible values are "READY", "ASLEEP", "SUSPENDED", "ASLEEP/SUSP", and "BROKEN".										
pppp	This is a temporary, hexadecimal priority that is sometimes assigned to the task by the Nucleus. This is done to improve system performance.										
qqqq	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.										
rrrr	This is the base, in hexadecimal, of the start address of the task's exception handler.										
ssss	This is the hexadecimal offset, relative to rrrr, of the start address of the task's exception handler.										
E ---	This option displays the principal attributes of the exchange whose token is represented by ITEM. The form of the display is as depicted in Figure 12-18 or 12-19, depending upon the type of the exchange.										

DEBUGGER

----- iRMX86 MAILBOX REPORT -----			
MAILBOX TOKEN	bbbb	CONTAINING JOB	gggg
# TASKS WAITING	cccc	# OBJECTS WAITING	hhhh
FIRST WAITING	dddJ/eeeef	QUEUE DISCIPLINE	iiiiiii
NAME(S)	aaaaaaaaaaaa	aaaaaaaaaaaa	aaaaaaaaaaaa
	aaaaaaaaaaaa	aaaaaaaaaaaa	aaaaaaaaaaaa

Figure 12-18. An iRMX 86™ Mailbox Report

The following describes the fields in Figure 12-18:

Field	Meaning
aaaaaaaaaaaa	Each such field can contain 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 printed here.
bbbb	This is a hexadecimal token for the mailbox.
cccc	This is the number, in hexadecimal, of tasks in the mailbox's task queue.
ddd	This is a 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	This is a 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	This is a 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

DEBUGGER

Field	Meaning
gggg	This is a hexadecimal token for the job containing the mailbox.
hhhh	This is the number, in hexadecimal, of objects in the mailbox's object queue.
iiiiiii	This describes the manner in which waiting tasks are queued in the mailbox's task queue. The possible values are FIFO and PRIORITY.

```

----- iRMX86 SEMAPHORE REPORT -----
SEMAPHORE TOKEN      bbbb          CONTAINING JOB      gggg
CURRENT VALUE        cccc          MAXIMUM VALUE       hhhh
# TASKS WAITING      dddd          QUEUE DISCIPLINE   iiiiii
FIRST WAITING        eeeeJ/ffffT

NAME(S)  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa
         aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa  aaaaaaaaaa
    
```

Figure 12-19. An iRMX 86™ Semaphore Report

The following describes the fields in Figure 12-19:

Field	Meaning
aaaaaaaaaaaa	Each such field can contain 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 printed here.
bbbb	This is a hexadecimal token for the semaphore.
cccc	This is the number, in hexadecimal, of units currently in the custody of the semaphore.
dddd	This is the number, in hexadecimal, of tasks waiting in the queue.
eeee	This is a hexadecimal token for the containing job of the first waiting task. It is absent if no tasks are waiting.
ffff	This is a hexadecimal token for the first waiting task. It is 0000 if no tasks are waiting.
gggg	This is a hexadecimal token for the semaphore's containing job.

DEBUGGER

Field	Meaning
hhhh	This is the maximum allowable number, in hexadecimal, of units that the semaphore may have in its custody.
iiiiiii	This describes the manner in which waiting tasks are queued in the semaphore's task queue. The possible values are FIFO and PRIORITY.
G ---	This option displays the principal attributes of the segment whose token is represented by ITEM. The form of the display is depicted in Figure 12-20.

----- iRMX86 SEGMENT REPORT -----				
SEGMENT TOKEN	bbbb	CONTAINING JOB		dddd
SEGMENT BASE	cccc	SEGMENT LENGTH		eeee
NAME(S)	aaaaaaaaaaaa	aaaaaaaaaaaa	aaaaaaaaaaaa	aaaaaaaaaaaa
	aaaaaaaaaaaa	aaaaaaaaaaaa	aaaaaaaaaaaa	aaaaaaaaaaaa

Figure 12-20. An iRMX 86™ Segment Report

The following describes the fields in Figure 12-20:

Field	Meaning
aaaaaaaaaaaa	Each such field can contain 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 printed here.
bbbb	This is a hexadecimal token for the segment.
cccc	This is the base address of the segment.
dddd	This is a hexadecimal token for the job that contains the segment.
eeee	This is the number, in hexadecimal, of bytes in the segment.

Viewing iRMX 86 System Lists --- The V Command

Syntax

The syntax for the V command is given in Figure 12-21.

DEBUGGER

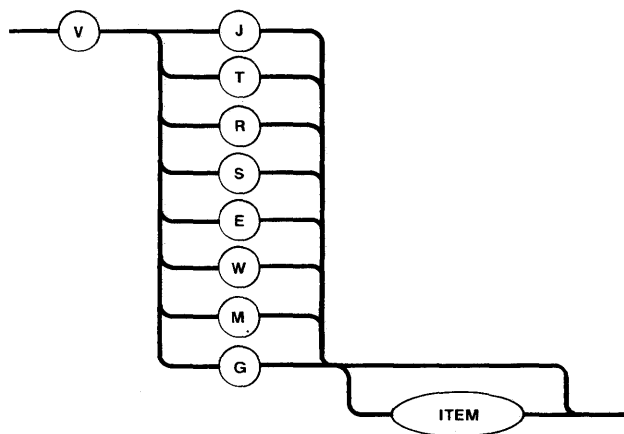


Figure 12-21. Syntax Diagram for Viewing iRMX 86™ System Lists

Explanation of V Command Options

J --- This option lists all jobs as

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

where

jjjj A job token.

pppp A token of its parent job. If the optional ITEM is included in the command and it contains a job token, then tokens of all of that job's children are listed. If the job designated by jjjj is the root job, then "ROOT" replaces "ppppJ".

T --- This option lists all tasks as

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

where

tttt A task token.

jjjj A token for the job that contains the task. If the optional ITEM is included in the command and it contains a job token, then tokens of all the tasks in that job are listed. An asterisk following a task token indicates that the task has overflowed its stack.

DEBUGGER

R --- This option lists all ready tasks as

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

where

tttt A token of a ready task.

jjjj A token for the job containing the task. If the optional ITEM is included in the command and it contains a job token, then tokens of all the ready tasks in that job are listed. An asterisk following a task token indicates that the task has overflowed its stack.

S --- This option lists all suspended tasks as

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

where

tttt A token of a suspended task.

jjjj A token for the job containing the task. If the optional ITEM is included in the command and it contains a job token, then tokens of all the suspended tasks in that job are listed. An asterisk following a task token indicates that the task has overflowed its stack.

E --- This option lists all exchanges as

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

where

xxxx A token for an exchange.

t The type of the exchange (M for mailbox or S for semaphore).

jjjj A token for the job containing the exchange. If the optional ITEM is included in the command and it contains a job token, then tokens of all the exchanges in that job are listed.

W --- This option lists the task queues at exchanges as

DEBUGGER

```
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 A token for an exchange

t The type of the exchange (M for mailbox, S for semaphore).

tttt A token for a task which is queued at that exchange.

jjjj A token for the job containing the task. If the optional ITEM is included in the command and it contains the token for an exchange, then tokens for the tasks in that exchange's task queue are listed. An asterisk indicates that the task has overflowed its stack.

M --- This option lists the object queues at mailboxes as

```
ML jjjjJ/mmmM = jjjjJ/ooooT jjjjJ/ooooT ... jjjjJ/ooooT
ML jjjjJ/mmmM = jjjjJ/ooooT jjjjJ/ooooT ... jjjjJ/ooooT
```

```
·
·
·
```

```
ML jjjjJ/mmmM = jjjjJ/ooooT jjjjJ/ooooT ... jjjjJ/ooooT
```

where

mmm A token for a mailbox.

oooo A token for an object in that exchange's object queue.

t The type of the object (J for job, T for task, M for mailbox, S for semaphore, and G for segment).

jjjj A token for the job containing the exchange or object. If the optional ITEM is included in the command and it contains the token for a mailbox, then tokens for the objects in that mailbox's object queue are listed.

DEBUGGER

G --- This option lists the segments as

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

where

gggg	A token for a segment.
jjjj	A token for the job containing the segment. If the optional ITEM is included in the command and it contains the token for a job, then tokens for the segments in that job are listed.

Exiting the Debugger --- The Q Command

Syntax

The syntax for the Q command is given in Figure 12-22.

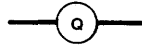


Figure 12-22. Syntax Diagram for Exiting the Debugger

Effects

This command deactivates the Debugger. When a debugging session is terminated, the tables and lists the Debugger maintains are not destroyed. Q also causes the message "EXIT iRMX 86 DEBUGGER" to be displayed.

USING SYMBOLIC NAMES WHILE DEBUGGING

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. Commands for defining, changing, listing, and deleting numeric variables are described later in this section.
- The object directory of the current job. The current job is defined to be the job that contains the breakpoint task. If there is no breakpoint task, the current job is the root job.
- The object directory of the root job.

DEBUGGER

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.

Defining a Numeric Variable --- The D Command

Syntax

The syntax for the D command is given in Figure 12-23.

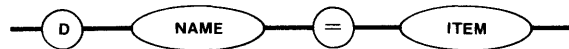


Figure 12-23. Syntax Diagram for Defining a Numeric Variable

Effects

This command puts the NAME and the value of ITEM in the Debugger's symbol table.

Changing a Numeric Variable

Syntax

The syntax for this command is given in Figure 12-24.

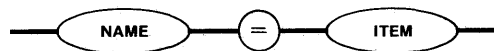


Figure 12-24. Syntax Diagram for Changing a Numeric Variable

Effects

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

Deleting a Numeric Variable --- The Z Command

Syntax

The syntax for the Z command is given in Figure 12-25.



Figure 12-25. Syntax Diagram for Deleting a Numeric Variable

Effects

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

Viewing Numeric Variables --- The L Command

Syntax

The syntax for the L command is given in Figure 12-26.

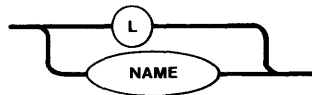


Figure 12-26. Syntax Diagram for Viewing Numeric Variables

Effects

The L command causes all numeric variable names and their associated values to be listed. If only NAME is specified, only one pair is listed. In either case, one pair is listed per line in the format

NAME=xxxx

where xxxx is the associated value.

APPENDIX A. iRMX 86™ DATA TYPES

The following are the data types that are recognized by the iRMX 86 Operating System:

BYTE - An unsigned, one byte, binary number.

WORD - An unsigned, two byte, binary number.

INTEGER- A signed, two byte, binary number that is stored in two's complement form.

OFFSET - A word whose value represents the distance from the base of a segment.

TOKEN - A word whose value identifies an object.

POINTER- Two words containing the base of a segment and an offset, in the reverse order.

STRING - A sequence of consecutive bytes. The first byte contains the number (not to exceed 12) of bytes that follow it in the string.

APPENDIX B. iRMX 86™ TYPE CODES

Each iRMX 86 object type is known within iRMX 86 systems by means of a numeric code. For each code, there is a mnemonic name that can be substituted for the code. The following lists the types with their codes and associated mnemonics.

<u>OBJECT TYPE</u>	<u>INTERNAL MNEMONIC</u>	<u>NUMERIC CODE</u>
Job	T\$JOB	1
Task	T\$TASK	2
Mailbox	T\$MAILBOX	3
Semaphore	T\$SEMAPHORE	4
Segment	T\$SEGMENT	6

INDEX

allocation of memory 6-1, 6-2
asleep state 2-2, 4-1
asleep-suspended state 2-2, 4-1
baud rate 11-5
breakpoint 12-7
 exception 12-8, 12-10
 exchange 12-8, 12-9
 execution 12-8, 12-9
breakpoint list 12-8, 12-19
breakpoint request 12-9
breakpoint state 12-9
breakpoint task 12-11, 12-17, 12-19, 12-20, 12-21, 12-23
breakpoint variable 12-9
carriage return character 11-2, 11-4, 12-3
CATALOG\$OBJECT 4-3, 7-2, 10-5
child job 3-1, 6-3
Command Dictionary 10-2
communication between tasks 5-1
condition code 8-1, 8-3
constant in Debugger 12-5
control-C command 11-2, 11-5
control-D command 12-2
control-O command 11-2, 11-5
control-Q command 11-2, 11-5, 12-3
control-R command 11-2, 11-3
control-S command 11-2, 11-4, 12-3
control-X command 11-2, 11-3
control-Z command 11-2, 11-4
CREATE\$JOB 3-3, 3-4, 10-7
CREATE\$MAILBOX 5-2, 10-13
CREATE\$SEGMENT 6-1, 6-4, 10-14
CREATE\$SEMAPHORE 5-4, 10-15
CREATE\$TASK 4-4, 10-17
current address 12-25
current display mode 12-24
current line 11-2, 12-3
current offset 12-25
current segment base 12-25
data types A-1
Debugger 1-1, 12-1
 as exception handler 8-2, 10-53, 12-2
 constant 12-5
 exiting 12-44
 expression 12-5
 invoking 12-2
 item 12-8
 name 12-5
 symbolic names 12-44
 syntax 12-4

INDEX

DELETE\$JOB 3-4, 9-3, 10-20
DELETE\$MAILBOX 5-2, 10-22
DELETE\$SEGMENT 6-3, 6-4, 10-23
DELETE\$SEMAPHORE 5-4, 10-24
DELETE\$TASK 4-4, 9-3, 10-25
destination field 12-28
DISABLE 9-2, 9-10, 10-26
disabling interrupts 9-2, 9-10, 10-26
ENABLE 9-2, 9-10, 10-27
enabling interrupts 9-2, 9-10, 10-27
ENTER\$INTERRUPT 9-3, 9-4, 9-5, 9-6, 10-28
environmental condition 8-1, 8-3
escape (ESC) character 11-2, 11-4, 12-3
exception handler 2-5, 4-4, 8-1, 8-2, 10-29, 10-49
exception mode 4-4, 8-2
exceptional conditions 2-5, 8-1, 8-3
 programmer error 2-5, 8-1, 8-3
 environmental condition 2-5, 8-1, 8-3
exchange 5-1
 mailbox 5-1
 semaphore 5-3
execution state 2-2, 4-1
 asleep 2-2, 4-1
 asleep-suspended 2-2, 4-1
 ready 2-2, 4-2
 running 2-2, 4-2
 suspended 2-2, 4-1
 transitions between states 4-2
EXIT\$INTERRUPT 9-3, 9-4, 9-6, 9-10, 10-30
expression in Debugger 12-5
GET\$EXCEPTION\$HANDLER 8-4, 10-31
GET\$LEVEL 9-3, 9-10, 10-33
GET\$POOL\$ATTRIB 6-2, 6-4, 10-34
GET\$PRIORITY 4-5, 10-36
GET\$SIZE 6-4, 10-37
GET\$TASK\$TOKENS 3-3, 4-5, 7-2, 10-38
GET\$TYPE 7-1, 7-2, 10-39
ICE-86 In-circuit Emulator 12-1, 12-2
input request mailbox 11-6
input request message 11-7, 11-8, 11-9
interrupt 9-1
 handler 2-5, 9-3, 9-4, 9-5, 9-8, 9-9, 10-28,
 10-30, 10-47, 10-54, 10-59, 10-63
 level 9-1, 9-2, 9-5, 9-7, 9-10, 10-26, 10-27, 10-33
 task 9-3, 9-4, 9-5, 9-8, 9-10, 10-47, 10-54, 10-63
 vector 9-1
 vector table 9-1
invoking the Debugger 12-2
I/O System 2-1, 11-1
iSBC 86/12A Monitor 12-1
item in Debugger 12-8

INDEX

job 2-1, 2-3, 3-1, 10-7, 10-20, 10-42
 child 3-1
 memory pool 3-1, 3-2, 3-3, 6-1, 6-2, 6-3
 object directory 2-4, 3-1, 7-1
 object limit 3-1
 parameter object 3-3, 4-5
 parent 3-1
 pool size 3-3, 6-1, 6-2
 task limit 3-1
 tree 2-3, 3-1
level 9-1, 9-2, 9-5, 9-7, 9-10, 10-26, 10-27, 10-33
level 7 interrupts 9-7
line feed character 11-2, 11-4, 12-3
LOOKUP\$OBJECT 4-2, 7-1, 7-2, 10-40
mailbox 2-1, 2-4, 5-1, 10-13, 10-22, 10-43, 10-49
memory 2-4, 3-1, 3-3, 6-1, 10-14, 10-23, 10-34, 10-37, 10-58
 allocating 2-4, 6-3, 10-14, 10-23
 available 6-2
 borrowing 3-3, 6-2, 10-14
 maximum pool size 6-2
 minimum pool size 6-2, 10-54
mutual exclusion 2-4, 5-3
name in Debugger 12-5
normal character 11-1, 11-2
normal mode 11-4
Nucleus 1-1, 2-1
object 2-1, 7-1, 10-5, 10-39, 10-40, 10-62
 job 2-1, 2-3, 3-1
 mailbox 2-1, 2-4, 5-1
 segment 2-1, 2-4, 6-1
 semaphore 2-1, 2-4, 5-3
 task 2-1, 2-2, 4-1
object directory 2-4, 3-1, 3-3, 7-1, 10-5, 10-40, 10-62
object queue 5-1
object type 2-1, 7-1
OFFSPRING 3-3, 3-4, 10-42
output request mailbox 11-6
output request message 11-7
output-only Terminal Handler 11-1
parameter object 3-3, 4-5, 10-7, 10-38
parent job 3-1
priority 2-2, 4-1, 4-2, 5-2, 5-3, 9-2, 9-5, 9-7, 10-36
programmer error 8-1, 8-3
queue 5-1, 5-2, 5-3
 first-in-first-out 5-1, 5-2, 5-3
 priority 5-1, 5-2, 5-3
queueing mode 11-4

INDEX

ready state 2-2, 4-2
 RECEIVE\$MESSAGE 4-2, 5-2, 10-43
 RECEIVE\$UNITS 4-2, 5-3, 10-45
 request message 11-7
 RESET\$INTERRUPT 9-3, 9-10, 10-47
 RESUME\$TASK 4-4, 4-5, 10-48
 RMX/86 Operating System 1-1
 root job 2-3, 3-1, 4-5, 7-2, 10-38
 RQNORMIN 11-6, 11-10
 RQNORMOUT 11-4, 11-5, 11-6, 11-7
 RQRATE 11-5
 RQ\$ABORT\$AP 11-5
 rubout 11-2, 11-3
 running state 2-2, 4-2
 segment 2-1, 2-4, 6-1, 10-14, 10-23, 10-34, 10-37, 10-58
 semaphore 2-1, 2-4, 5-3, 10-15, 10-24, 10-45, 10-51
 semaphore limit 5-3, 10-15, 10-51
 SEND\$MESSAGE 4-3, 5-2, 10-49
 SEND\$UNITS 4-3, 5-3, 10-51
 SET\$EXCEPTION\$HANDLER 8-2, 8-4, 10-52
 SET\$INTERRUPT 9-3, 9-4, 9-5, 9-10, 10-54
 SET\$POOL\$MINIMUM 6-2, 6-4, 10-58
 SIGNAL\$INTERRUPT 9-3, 9-4, 9-5, 9-10, 10-59
 SLEEP 4-5, 10-60
 special character 11-1, 11-2
 suppression mode 11-4
 SUSPEND\$TASK 4-3, 4-4, 10-61
 suspended state 2-2, 4-1
 suspension depth 4-1
 symbolic names for debugging 12-44
 synchronization 5-1, 5-3
 syntax in Debugger commands 12-4
 system call 2-1
 system clock 9-2
 system exception handler 8-2
 task 2-1, 2-2, 4-1, 10-17, 10-25, 10-36, 10-38, 10-48, 10-60,
 10-61
 arbitration algorithm 2-2, 4-2
 exception handler 2-5, 4-4, 8-2
 interrupt 9-3, 9-4, 9-6, 9-8
 limit 3-1
 Nucleus' view 2-2
 priority 2-2, 4-1, 4-2, 5-2, 5-3, 9-2, 9-5, 9-7, 10-36
 states 2-2, 4-1, 4-2
 suspension depth 4-1
 task queue 5-1, 5-2, 5-3
 Terminal Handler 1-1, 11-1
 token 2-1, 6-1
 tree of jobs 2-3, 3-1
 type 2-1, 7-1, A-1, B-1
 type code 7-1, B-1
 UNCATALOG\$OBJECT 7-2, 10-62
 WAIT\$INTERRUPT 9-3, 9-5, 9-10, 10-63



REQUEST FOR READER'S COMMENTS

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

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

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

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

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

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

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

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply.

WE'D LIKE YOUR COMMENTS . . .

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



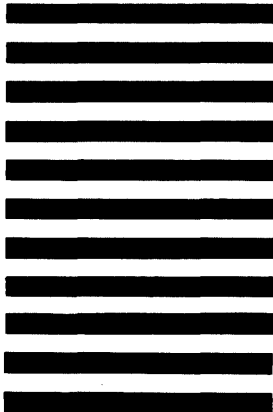
**NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES**

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

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
3585 S.W. 198th
Aloha, Oregon 97005

O.M.S. Technical Publications





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

Printed in U.S.A.