RadiSys.

iRMX® Programming Concepts for DOS

RadiSys Corporation
5445 NE Dawson Creek Drive
Hillsboro, OR 97124
(503) 615-1100
FAX: (503) 615-1150
www.radisys.com
07-0571-01
December 1999

Quick Contents

Notational Conventions

Most of the references to system calls in the text and graphics use C syntax instead of PL/M (for example, the system call **send_message** instead of **send$message**). If you are working in C, you must use the C header files, *rmx_c.h*, *udi_c.h* and *rmx_err.h*. If you are working in PL/M, you must use dollar signs ($) and use the *rmxplm.ext* and *error.lit* header files.

This manual uses the following conventions:

- Syntax strings, data types, and data structures are provided for PL/M and C respectively.
- All numbers are decimal unless otherwise stated. Hexadecimal numbers include the `H` radix character (for example, `0FFH`). Binary numbers include the `B` radix character (for example, `11011000B`).
- Bit 0 is the low-order bit. If a bit is set to 1, the associated description is true unless otherwise stated.
- `Data structures and syntax strings appear in this font.`
- **System call names and command names appear in this font.**
- PL/M data types such as BYTE and SELECTOR, and iRMX data types such as STRING and SOCKET are capitalized. All C data types are lower case except those that represent data structures.
- The following OS layer abbreviations are used. The Nucleus layer is unabbreviated.

AL	Application Loader
BIOS	Basic I/O System
EIOS	Extended I/O System
HI	Human Interface
UDI	Universal Development Interface

- Whenever this manual describes I/O operations, it assumes that tasks use BIOS calls (such as **rq_a_read**, **rq_a_write**, and **rq_a_special**). Although not mentioned, tasks can also use the equivalent EIOS calls (such as **rq_s_read**, **rq_s_write**, and **rq_s_special**) or UDI calls (**dq_read** or **dq_write**) to do the same operations.

Contents

Tables

Figures

Introduction 1

This manual discusses the programming concepts necessary to produce real-time applications for an environment that includes DOS and the iRMX® OS.

This manual is for programmers who are familiar with:

- Applications programming in the DOS and Windows environment
- Terms and concepts for the iRMX OS

  See also: Introducing the iRMX Operating Systems, System Concepts

- C or PL/M programming language

  See also: *iC-386 Compiler User's Guide*, *PL/M-386 Programmer's Guide*

The DOSRMX OS provides a set of powerful extensions to DOS. With it you can develop DOS applications that incorporate the preemptive, priority-based multitasking and real-time response of the iRMX OS.

DOSRMX enables:

- MS-DOS or PC-DOS OSs to run concurrently with the iRMX OS on the same microprocessor and to share the same console
- Existing DOS Real Mode application programs, including most off-the-shelf applications, to run under DOS with no modification
- Existing iRMX application programs to run under DOSRMX with no modification, while maintaining real-time performance
- DOS application programs to make iRMX Nucleus system calls and to communicate directly with 32-bit Protected Mode iRMX application programs
- iRMX application programs to make DOS and ROM BIOS system calls from within an iRMX task
- DOS programs to access iRMX files and iRMX programs to access DOS files
- Preconfigured and loadable file and device drivers and system jobs
- Simultaneous access to network services from DOS and the iRMX OS

Understanding the Environments

Some of the facilities discussed in this manual can be used in two environments. One environment consists of DOSRMX and DOS within a single system. The other environment consists of networked systems where one system can be any iRMX OS communicating with a system running DOS/Windows.

These facilities include Remote File Access. Using the NetBIOS interface to the OpenNET networking protocol and a standard DOS Network Redirector, DOS applications can access the iRMX file systems on a local DOSRMX system or remote systems running any iRMX OS.

Note

When running DOS on a stand-alone system, the PCLINK2 Networking Adapter facilitates access to the OpenNET networking protocol. This access is restricted to systems running only Windows 3.1 or 3.11.

Table 1-1 shows the facilities required to perform certain operations between the iRMX OS and Windows.

Table 1-1. Facilities for Supporting Various iRMX OS and Windows Configurations

	Windows PC, iRMX System, TCP/IP network	**Windows PC, iRMX System, ISO network**	**DOSRMX, one PC**
File Access	NFS*, FTP*	NetBIOS (iRMX-NET)	NetBIOS (iRMX-NET)
Virtual Terminal Support	Telnet*		

*See also: *TCP/IP and NFS for the iRMX Operating System*

Running DOS and the iRMX® OS on the Same System

By itself, DOS does not use the advanced features of the Intel386 , Intel486 and Pentium® microprocessors such as Protected Mode Addressing.

The iRMX OS exploits more of the features of these microprocessors, allowing tasks to run concurrently and to reside in up to 4 Gbytes of memory. Under the iRMX OS, these microprocessors also run in 32-bit Protected Mode.

DOSRMX encapsulates DOS as an iRMX task and runs that task in Virtual 86 Mode (VM86). The encapsulated DOS task (DOS and its application programs) runs in the first 1 Mbyte of memory, and the iRMX OS runs in the remaining memory.

A terminate and stay resident (TSR) program, *rmxtsr.exe*, provides a small buffer in DOS memory which enables DOS and the iRMX OS to exchange data.

See also: Loadable system jobs, *System Configuration and Administration*,

VM86 Dispatcher

The VM86 Dispatcher enables DOS to run as a task under DOSRMX by:

- Switching the microprocessor addressing mode, depending on which OS is running
- Ensuring that interrupts are handled by the appropriate OS
- Providing file sharing between the OSs
- Preventing hardware resource conflicts between the OSs

The VM86 Dispatcher is preconfigured into DOSRMX as an iRMX first-level system job.

See also: System jobs, *System Concepts*

DOS is not supplied with iRMX II.2.3; you must install PC-DOS or MS-DOS OS before you can run DOSRMX. You can install compatible off-the-shelf DOS applications before you run DOSRMX.

VM86 Protected Mode Extensions

The VM86 Protected Mode extensions allow you to use Protected Mode services provided by the iRMX OS. Using these extensions, DOS application programs running in VM86 mode can access Protected Mode services such as 4 Gbyte addressing, as well as the iRMX system calls.

The Intel-supplied VM86 Protected Mode extensions provided by the VM86 Dispatcher include:

- DOS Real-time extension (RTE)
- Network Redirector (NETRDR)

You can also write your own VM86 Protected Mode extensions.

See also: **rqe_set_vm86_extension**, in this manual and *System Call Reference*

Real-time Extension

The DOS Real-time Extension (RTE) enables you to call some of the iRMX Nucleus system calls from within a DOS application program. By using the RTE, a DOS application program can communicate with a concurrently-running iRMX application program using standard iRMX techniques. The RTE includes system calls that create and delete iRMX objects and descriptors, read and write segments, and catalog and look up objects.

Figure 1-1 illustrates how a DOS application makes a Nucleus system call.

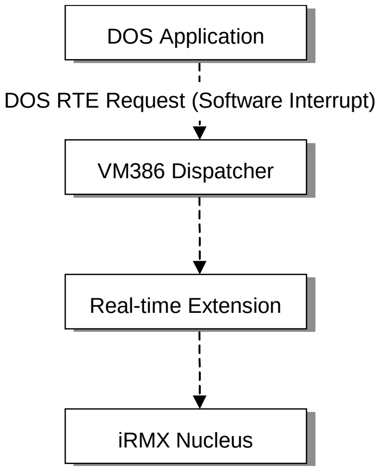

Figure 1-1. Making Nucleus System Calls with the DOS Real-time Extension

For example, the DOS application program may send or receive messages or data using a mailbox created by an iRMX application program. Similarly, an iRMX program may send or receive messages using a mailbox created by the DOS application program.

See also: DOS RTE, in this manual,
DOS-specific system calls, *System Call Reference*

Making DOS/ROM BIOS System Calls from the iRMX OS

DOSRMX enables iRMX programs to use the DOS and ROM BIOS software interrupt services, including any special ROM BIOS functions provided by add-in adapters, rather than implementing the same function as iRMX system calls.

Figure 1-2 illustrates how iRMX programs can make a DOS/ROM BIOS call.

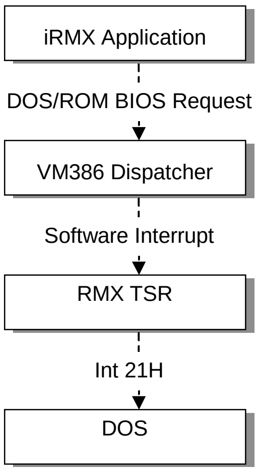

Figure 1-2. Making DOS and ROM BIOS Requests from an iRMX Application

For example, your iRMX application program can use the DOS **Get Free Disk Space** example to check available space on a network disk drive.

See also: Get Free Disk Space Example, Chapter 4,
rqe_dos_request, *System Call Reference*

File Access

The DOS and iRMX file systems are inherently different. However, a file driver provided with the iRMX OS allows DOS and iRMX application programs to share files.

The Encapsulated DOS (EDOS) file driver enables iRMX application programs to access files on a DOS partition, storage device, or network drive that has a drive letter mapped to it. EDOS allows both DOS and iRMX partitions to store both DOS and iRMX files.

Applications on a DOS partition can also access files on an iRMX partition by using a network job and these configuration files:

- The PCNET NetBIOS driver, *pcnet.exe* (installed on the DOS partition).
- The network redirector job, *netrdr.job* (loaded on the iRMX partition).
- The iNA 960 networking job, *i*.job*, appropriate to your OS and the iRMX-NET client and server jobs (*remotefd.job* and *rnetserv.job*).

  See also: Using iRMX-NET in a DOS Environment, *System Configuration and Administration*

Under DOSRMX, you can choose whether to use only the DOS file system, or a partitioned file system including a DOS partition and one or more iRMX partitions. At least one DOS drive is required in an DOSRMX system.

See also: File access, *Command Reference*,
Installing DOSRMX, *Installation and Startup*

Figure 1-3 illustrates how DOS file requests are carried out by the I/O System.

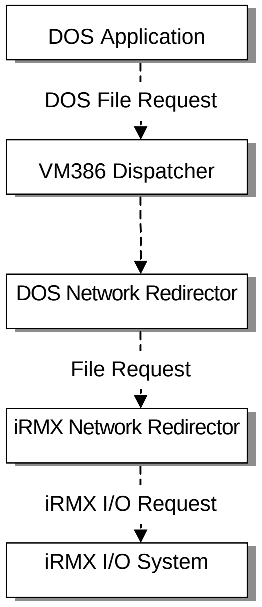

Figure 1-3. Using Networking to Access Files on the iRMX File System

The iRMX file system could be a separate drive or an iRMX partition on a drive containing both DOS and iRMX partitions, or even a remote drive accessed through iRMX-NET.

See also:	File types, *System Concepts*

EDOS File Driver

The EDOS file driver uses DOS as a file server to access DOS files. It maps iRMX file driver interfaces to DOS system calls. Therefore, files on the DOS drives appear to the iRMX application just as they would on an iRMX drive.

See also: **attachdevice** command, *Command Reference*

Figure 1-4 illustrates how iRMX applications make DOS file requests.

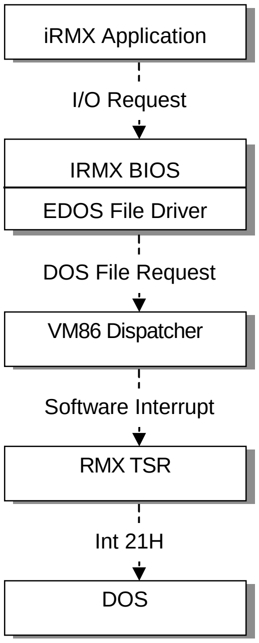

Figure 1-4. Accessing DOS Files with the EDOS File Driver

Networking

DOSRMX supports both DOS and iRMX networking. This support provides a variety of capabilities:

- DOS and iRMX applications that communicate on the network run unchanged when they run within the same system.
- DOS files can be accessed from a remote file consumer without a dedicated file server.

- DOS and iRMX OSs running within the same system can share a single network controller.
- OpenNET networking support provides connections to computers running the DOS and UNIX OSs.
- DOS networking applications can use the network controller in a Multibus I or Multibus II system.

A PC running DOSRMX can run this network software:

- MS-Net client or server
- IBM PC LAN client or server
- Novell NetWare client
- Combinations of MS-Net and NetWare on one computer

PCs running DOSRMX can also run this iRMX network software:

- iRMX-NET consumer and server for remote file access, which can coexist with DOS network software
- iNA 960 jobs for a programmatic interface, but which cannot coexist with DOS network software.
- Null data link network jobs that allow DOS to access the iRMX file system without an Network Interface Controller (NIC).

See also: Network jobs, *i*.job, System Configuration and Administration*, Introduction, *Network User's Guide and Reference*

File and Device Drivers

The DOSRMX software includes preconfigured file drivers and device drivers that can be loaded dynamically.

Loadable File and Device Drivers

These driver allow you to write procedures to invoke and interface to additional custom, random access, and terminal hardware.

See also: Loadable file and device drivers, *Driver Programming Concepts* and *System Configuration and Administration*

System Configuration

DOSRMX is preconfigured to run in the DOS ndows environment, however you may change some aspects of the OS for a particular application.

Certain parts of the OS are loadable, including loadable file and device drivers and loadable jobs.

You load these elements into the system with the **sysload** command in the *:config:loadinfo* file. Loadable device drivers allow you to write procedures to invoke and interface to additional custom, random access, and terminal hardware. Loadable file drivers enable you to include custom file drivers.

The OS includes an *rmx.ini* file for load-time configuration. As layers of the OS boot, they read entries from this file. The *rmx.ini* file contains entries that match settings preconfigured into DOSRMX. You can modify the existing entries to fine-tune your use of the OS.

You can also use the Interactive Configuration Utility (ICU) to change the configuration of the DOSRMX or iRMX for PCs OSs.

See also: Loadable jobs and drivers, *System Configuration and Administration*,
Loadable device drivers, *Driver Programming Concepts*,
Physical device names, *Command Reference,*
ICU Quick Reference, *ICU User's Guide and Quick Reference*

❑❑❑

DOS Real-Time Extension 2

The DOS Real-time Extension (RTE) enables DOS application programs to use the real-time protected mode features of the iRMX Nucleus. Not all iRMX Nucleus system calls are supported by the RTE.

RTE System Calls

A complete list of calls that are supported by both DOS and the iRMX OS is given in Table 2-1, which also lists the function code value for each RTE system call.

There are two additional calls that are only available from DOS. These calls, **rqe_read_segment** and **rqe_write_segment**, allow the application program to transfer data between DOS in VM86 memory and the iRMX OS in Protected Memory.

See also: **rqe_read_segment** and **rqe_write_segment** calls, *System Call Reference*

Table 2-1. RTE System Calls

Code	Call Name	Description
0	**create_mailbox**	Create an object or data mailbox
1	**delete_mailbox**	Delete an object or data mailbox
2	**send_message**	Send an object to a mailbox
3	**send_data**	Send data to a mailbox
4	**receive_message**	Receive an object
5	**receive_data**	Receive data
6	**create_semaphore**	Create a semaphore
7	**delete_semaphore**	Delete a semaphore
8	**send_units**	Send a unit to a semaphore
9	**receive units**	Receive a unit from a semaphore
10	**create_region**	Create a region
11	**delete_region**	Delete a region
12	**send_control**	Release control of a region
13	**receive_control**	Receive control of a region
14	**accept_control**	Accept control of a region
15	**create_segment**	Create a segment
16	**delete_segment**	Delete a segment
17	**get_size**	Get the size of a segment
18	**rqe_get_address**	Get the physical address of a segment
19	**rqe_create_descriptor**	Create a PVAM descriptor
20	**rqe_delete_descriptor**	Delete a PVAM descriptor
21	**rqe_change_descriptor**	Change a PVAM descriptor
22	**catalog_object**	Catalog an object
23	**uncatalog_object**	Uncatalog an object
24	**lookup_object**	Lookup an object
26	**get_task_tokens**	Get task or job token
27	**get_type**	Get the type of an object
28	**sleep**	Sleep for a specified time
30	**rqe_read_segment**	Read from PVAM to a Real Mode segment
31	**rqe_write_segment**	Write to a PVAM segment from a real mode segment

Generally, the RTE system calls allow the DOS application program to manipulate iRMX objects, semaphores, mailboxes, regions, segments and Protected Virtual Address Mode (PVAM) descriptors, and to communicate with iRMX tasks.

See also: VM86 Protected Mode Extensions, Chapter 3

If the DOS application program invokes an RTE system call that creates an iRMX object (such as a mailbox or PVAM descriptor), it must delete the iRMX object with the corresponding DOS RTE delete call from DOS. If the DOS application program does not explicitly delete the object, the object will be deleted upon termination of the DOS application program, or upon DOS being restarted.

The syntax and the semantics of the parameters for RTE system calls 0 to 28 are the same as the iRMX Nucleus system calls of the same name except for pointer parameters. The RTE system calls can return condition codes not returned by their Nucleus counterparts.

See also: System call descriptions, *System Call Reference*

All pointers parameters in these calls must be Real Mode pointers. Real Mode pointers consist of two 16-bit WORDs where the high WORD contains the base address of a 64 Kbyte segment and the low WORD contains the offset that points into the segment.

RTE is implemented by the VM86 Dispatcher in the DOS RTE system job. The DOS RTE job installs itself as a VM86 Protected Mode Extension at interrupt vector B8H. DOS RTE converts Real Mode pointers to PVAM pointers.

RQEGetRmxStatus Call

Use the **RQEGetRmxStatus** RTE call to check if the iRMX OS is loaded. Use this call before any other RTE calls to insure RTE services are available. Unpredictable results occur if RTE calls are called when iRMX is not present.

See also: **RQEGetRmxStatus**, *System Call Reference*

The call returns `E_OK` if iRMX is loaded and running, or `E_EXIST` if iRMX is not present or unavailable. The call is provided in binary form as a linkable module in the file *\rmx386\demo\rte\lib\rmxfuncs.obj*.

RTE Files

The C header file *\rmx386\demo\rte\lib\rmxintfc.c* contains all the declarations for the RTE functions.

When developing DOS applications which make RTE calls, include the file *rmxintfc.h* and compile with the /AL switch (for Microsoft C). Use this file *rmxintfc.h* with all models of compilation. The file *rmxc.h* is specific to the RTE demo; do not use it in other applications.

There are three versions of the DOS RTE libraries: *dosrtec.lib* for compact model compilations, *dosrtes.lib* for small model compilations, and *dosrtel.lib* for large model compilations. The *\rmx386\demo\rte\lib* directory contains source for the libraries.

See also: *\rmx386\demo\rte\lib\readme.txt* file, for more information about the source code, header files, and libraries

RTE Objects Limitation

The RTE job in DOSRMX maintains a table of all objects created by it on behalf of DOS applications. It handles up to 512 RTE-created iRMX objects, which is sufficient for most applications. However, this limit may be reached accidentally. When an RTE-created object is deleted in an iRMX task, the entry from the RTE table still remains. This causes the table to fill up with deleted objects.

The solution is to create a mailbox and send to it the tokens of objects to delete. Then have code in your DOS application which, when waiting for an iRMX event, queries the iRMX mailbox for objects to be deleted. If you delete all RTE objects that occur (which means iRMX applications receiving RTE-created objects need to send them to this mailbox for deletion), the RTE object table does not fill up and should then be able to handle all the active RTE objects needed by an application.

Making an RTE System Call

All RTE system calls are accessed using a single software interrupt. The microprocessor registers and the Real Mode stack are used for passing the parameters of the RTE system call. One of the parameters passed is the RTE function code.

To invoke an RTE system call, the DOS application program must perform these actions:

1. Push all the parameters required by the RTE system call onto the Real Mode stack using the PL/M-286 convention. That is, the first parameter is pushed onto the stack, followed by the second and subsequent parameters.
2. Load the SI register to point to the last parameter pushed onto the Real Mode stack.
3. Load the AX register with the desired RTE function code.
4. Generate the RTE software interrupt request number, B8H.

This causes the RTE system call to execute and return control to DOS. When the DOS application program resumes, it must clear the parameters used by the RTE system call from the Real Mode stack.

If the DOS RTE system call returns a WORD (16-bit), it will be placed in the AX register. If it returns a DWORD, the high WORD will be placed in the DX register and the low WORD will be placed in the AX register.

DOS and its application programs run as an iRMX task under DOSRMX. If the application programs invoke RTE functions, they must obey the normal iRMX rules of not invoking the RTE from a hardware interrupt handler. In particular, the DOS TSR programs that typically hook themselves onto the hardware clock or keyboard interrupts must not issue RTE calls.

Using RTE Functions

This example illustrates how a DOS application program can invoke one of the RTE functions. This example uses **rq_create_mailbox**. The example was compiled using Microsoft's assembler, MASM.

```
mov    ax,fifombx
push   ax                     ; PUSH 1st parameter - flags
mov    ax, SEG    exception   ; Get 2nd parameter into ES:AX
mov    es, ax                 ;
mov    ax, OFFSET exception   ;
push   es                     ; PUSH 2nd parameter -
push   ax                     ;    pointer to exception
mov    si,sp                  ; point si to last parameter
mov    ax,rqcreatembx         ; setup AX with FUNCTION CODE
int    0B8H                   ; CALL DOSRTE
add    sp,6                   ; remove parameters from stack
mov    mbx_tk,ax              ; save token
mov    ax,exception           ;
cmp    ax,eok                 ; check for validity
jne    error_p                ;
```

DOS RTE Demonstration

The DOS RTE demonstration program is menu-driven and enables you to exercise the RTE system calls at the DOS console. Two executable versions of the program are supplied: one runs as an iRMX application program, and the other runs as a DOS application program.

The source code and the executable for the demonstration programs are in the *\rmx386\demo\rte\obj\* directory. The executable has two parts:

- *demo* is the iRMX part
- *demo.exe* is the DOS part

Note
The DOS RTE demonstration program was compiled using Microsoft C, Version 7.0, compact model. If you are using the same compiler and model you can use the source as it is. Otherwise, compile the source using your compiler, make any necessary changes, and then recompile.

Both programs create, send, and delete iRMX objects, data, etc. The iRMX program makes iRMX system calls; the DOS program makes calls to the RTE system calls.

Examine the demonstration program source code to see how the RTE system calls are invoked.

With one exception, the iRMX and DOS programs share the same source code, which has been compiled conditionally. This demonstrates how you can create your own application programs to run under either the DOS or the iRMX OS, and subsequently port them between the OSs.

The RTE system calls **rqe_read_segment** and **rqe_write_segment** are demonstrated by the Data Transfer (Real Mode/PVAM) functions of two different demonstration programs. The DOS version performs these functions by making RTE system calls; the iRMX program has code written specifically for this operation. This is necessary since the **rqe_read_segment** and **rqe_write_segment** system calls provided by the RTE are not required for the iRMX OS.

See also: **rqe_read_segment** and **rqe_write_segment**, *System Call Reference*

Example: Running the Demonstration Program

To start the demonstration, change to the *\rmx386\demo\rte\obj\* directory. If you are at an iRMX prompt, run the iRMX *demo* program. If you are at a DOS prompt, run the DOS *demo.exe* program. Both programs display this menu:

```
DOS/iRMX Real Time Extensions Demo Program
========================================

1. Mailboxes (Objects) Functions
2. Mailboxes (data) Functions
3. Semaphore Functions
4. Segment Functions
5. Descriptor Functions
6. Data Transfer Functions
7. Display Help on above functions
8. Exit (terminate program)

Enter option (1 to 8) :-
```

Press the <Alt +> and <Alt -> keys (using the plus and minus keys on the numeric keypad) to change the background and foreground colors for the iRMX version of the demonstration. Since the appearance of the menus is identical, you can use color to tell you whether you are in the DOS or the iRMX version.

See also: Changing iRMX Console Color, *Installation and Startup*

You may invoke six different types of functions: mailboxes for data and objects, semaphores, PVAM segments, descriptors, and data display. The functions are described in the following sections.

Mailboxes (Objects) Functions

To invoke any of the Object Mailbox functions, enter:

```
1<CR>
```

in response to the Main Menu prompt. A menu similar to the Object Mailbox menu appears.

```
Object Mailbox Functions
=======================

1. Send object to mailbox
2. Receive object from mailbox
3. RETURN to previous menu

Enter option (1 to 3) :-
```

These functions allow you to send and receive objects (segments or descriptors) to or from a named mailbox. The mailbox may be created by either the DOS or iRMX version of the demonstration program.

Send Object to Mailbox

If the mailbox does not exist, it is created; if the named object does not exist, a segment is created for the object.

Receive Object from Mailbox

If the received object is a segment, its name is displayed. Otherwise, the iRMX token for the object is displayed, or if there are no objects, an E_TIME condition code is displayed.

If the mailbox does not exist, an error message appears.

Mailboxes (Data) Functions

To invoke any of the Data Mailbox functions, enter:

```
    2<CR>
```

in response to the Main Menu prompt. A menu similar to the Object Mailbox appears.

Data mailbox functions send and receive a string of text (up to a maximum of 127 characters) to and from a data type mailbox. The mailbox may be created by the DOS or iRMX version of this program.

Send Data to Mailbox

Enter the string at the prompt. The text entry must be terminated by a <CR>. If the requested data mailbox does not exist, one will be created.

Receive Data from Mailbox

This option receives a string of text from the specified data mailbox and displays the text and the size of the text string on the screen. If the specified mailbox does not exist, an error message appears.

Semaphore Functions

To invoke any of the Semaphore functions, enter:

```
3<CR>
```

in response to the Main Menu prompt. A menu similar to the Object Mailbox appears.

Semaphore functions send and receive units to and from a semaphore. The semaphore may be created by either the DOS or the iRMX version of this program.

Send Units to Semaphore

The semaphore will accept a maximum of 10 units. When prompted, enter the number of units to send.

Receive Units from a Semaphore

This option receives a requested number of units from a named semaphore and displays the remaining number of units at the semaphore. If the semaphore does not exist, an error message appears.

PVAM Segment Functions

To invoke any of the PVAM Segment functions, enter:

```
4<CR>
```

in response to the Main Menu prompt. This menu appears:

```
PVAM Segment Functions
=====================

1. Create PVAM Segment
2. Delete PVAM Segment
3. Display PVAM Segment
4. RETURN to previous menu

Enter option (1 to 4) :-
```

If you are not creating a segment, you can delete or display a segment created previously by either the DOS or the iRMX version of this program.

Create PVAM Segment

This option creates a named PVAM segment of any size. You can use this segment as either the source or destination of a copy operation to or from Real Mode memory. You can also pass the PVAM Segment to object mailboxes as well as display them by the Display PVAM Segment function.

Delete PVAM Segment

This option deletes a named PVAM segment or named descriptor. If you created the segment from DOS, delete it from DOS.

Display PVAM Segment

This option displays a PVAM segment or descriptor in blocks of 160 bytes maximum. The PVAM segment or descriptor displays in lines of 16 bytes, followed by the printable ASCII characters for each byte. If a byte is not a printable ASCII character, a . (period) is displayed instead. You are prompted for input to continue (any key) or quit (Q or q).

Descriptor Functions

To invoke any of the Descriptor functions, enter:

```
5<CR>
```

in response to the Main Menu prompt. A menu similar to the PVAM Segment menu appears.

If you are not creating a descriptor, you can delete or display a descriptor created previously by either the DOS or iRMX version of this program.

Create Descriptor

This option creates a named descriptor of any size and absolute address.

Delete Descriptor

If you created the descriptor from DOS, delete it from DOS.

Display Descriptor

This option displays a PVAM segment or descriptor in blocks of 160 bytes maximum. The PVAM segment or descriptor is displayed in lines of 16 bytes, followed by the printable ASCII characters for each byte. If a byte is not a printable ASCII character, a . (period) is displayed instead. You are prompted for input to continue (any key) or quit (Q or q).

The segment or descriptor is looked up under its user name.

Data Transfer Functions

To invoke any of the Data Transfer (Real Mode/PVAM) functions, enter:

```
6<CR>
```

in response to the Main Menu prompt. This menu appears:

```
REAL MODE/PVAM Copy Functions
============================

1. Copy PVAM segment to real mode address
2. Copy Real mode address to PVAM segment
3. RETURN to previous menu

Enter option (1 to 3) :-
```

Copy PVAM Segment to Real Mode Address

You are prompted for the Real Mode segment and offset.

⚠ **CAUTION**

Do not copy data over vital DOS system or application memory, or to memory mapped out to I/O devices. Otherwise, your system could develop problems.

Copy Real Mode Address to PVAM Segment

This option copies a specified Real Mode address to a specified PVAM Segment. You are prompted for the Real Mode Segment and Offset and also the PVAM Segment and Offset.

❑❑❑

VM86 Protected Mode Extensions 3

The VM86 Dispatcher enables you to write Protected Virtual Address Mode (PVAM) extensions for DOS. These extensions are also known as VM86 Protected Mode Extensions. These extensions allow DOS application programs running in VM86 Mode to change to Protected Mode, obtain Protected Mode services, and then return to VM86 Mode.

All VM86 Protected Mode Extensions are implemented as software interrupt handlers using the software interrupt instruction. The VM86 Dispatcher in Protected Mode intercepts all software interrupt requests. To run a VM86 Protected Mode Extension, the VM86 Dispatcher calls the required interrupt handler to service the particular request, and then returns to DOS in VM86 Mode. If the VM86 Dispatcher intercepts an interrupt request which is not a VM86 Protected Mode Extension request, that interrupt request is reflected back to DOS.

The RTE described in the previous chapter is an example of a VM86 Protected Mode Extension.

Installing VM86 Protected Mode Extensions

Each VM86 Protected Mode Extension you write, though implemented as an iRMX program, is invoked when a DOS application program issues an appropriate software interrupt. Each extension must be installed at a unique interrupt level and an extension may contain a number of subfunctions, as does the RTE. You can choose the method of passing the extension's subfunction. The RTE uses the AX register to hold the function's code.

Install the extension at its desired interrupt level using the **rqe_set_vm86_extension** system call.

See also: **rqe_set_vm86_extension**, *System Call Reference*

iRMX Interrupt Levels

Table 3-1 lists the interrupt levels in the Interrupt Descriptor Table (IDT) used by the DOSRMX OS.

Table 3-1. iRMX Interrupt Levels

Interrupt		Function
Hex	**Decimal**	
00H-10H	0-16	Microprocessor traps and DOS hardware vectors
11H-20H	17-32	*ROM BIOS services
21H-2FH	33-47	DOS services
38H-3FH	56-63	iRMX hardware vectors for Master PIC
50H-57H	80-87	iRMX hardware vectors for Slave PIC
5BH	91	Network Redirector
80H	128	Used by the VM86 Dispatcher
85H	133	iRMX Interface TSR, supports chaining however
B8H	184	DOS RTE
C3H	195	UDI

***** You may install extensions to monitor or evaluate these calls.

Interrupts and ranges not listed in Table 3-1 are available for user-written extensions.

To install an extension, call **rqe_set_vm86_extension** and pass it these parameters:

1. The desired interrupt level for the extension.
2. The entry point for the extension itself. This entry point defines where the extension is located in system memory so that it may be invoked when DOS makes the appropriate interrupt request.

3. The entry point for the extension's deletion handler. The deletion handler is not mandatory, but each extension can have one. Any extension which is used by DOS to create iRMX objects should have a deletion handler to delete those objects when the DOS program terminates.
4. A pointer to a WORD (16-bit) in system memory which the VM86 Dispatcher uses to return a status code for this call.

Once an extension has been installed, it remains active until it is deactivated with the **rqe_set_vm86_extension** system call. Call **rqe_set_vm86_extension** again and pass it the same parameters, but with the VM86 Extension Entry pointer set to null.

Extension Procedure Operation: DOS Interrupt Handling

Interrupts generated by DOS in VM86 mode are vectored to the PVAM handler referenced in the processor's IDT. The VM86 Dispatcher invokes a particular extension in response to an interrupt received at the `int_level` specified in the **rqe_set_vm86_extension** system call.

All DOS interrupts are intercepted by the VM86 Dispatcher and are processed as follows:

1. If an interrupt requires a Real Mode handler installed by DOS, the VM86 Dispatcher deflects the interrupt back to that Real Mode interrupt handler.
2. If the interrupt requires a PVAM interrupt handler, the Dispatcher enables the interrupt handler to run; the DOS application program is running in VM86 mode and all VM86 Mode-generated interrupts naturally vector to the PVAM interrupt handler in the IDT. The interrupt handler returns control back to the DOS application program upon termination.
3. If the interrupt requires a VM86 Extension, the VM86 Dispatcher calls the entry point of the extension. The extension then executes and returns to the VM86 Dispatcher, which then returns control back to the DOS application program that made the interrupt. The VM86 Dispatcher calls the VM86 Extension, and passes to it a pointer to a structure defining the DOS machine state and a value defining the context of the DOS interrupt handler. The VM86 Dispatcher expects the extension to return a byte indicating that the request has been processed completely.

   See also: **rqe_set_vm86_extension**, *System Call Reference*

Deletion Handler Operation

The VM86 Dispatcher calls all extension deletion handlers when any DOS program is deleted. Any of these conditions can delete a DOS program:

- When a DOS application program terminates using DOS system calls INT 20H or INT 21H
- When a <Ctrl-C> is typed in the middle of a DOS program, and the program has not changed the default <Ctrl-C> handler in DOS

All installed deletion handlers are called sequentially by the VM86 Dispatcher. The VM86 Dispatcher calls the deletion handler with a flag that indicates:

- If the current DOS program is being deleted
- If all DOS programs are being deleted

This helps the VM86 Dispatcher perform the appropriate cleanup. For example, if the VM86 extension has created iRMX objects, the deletion handler will know which objects to delete.

Note

Every DOS program has a unique identifier: the address of its Program Segment Prefix (PSP). The VM86 extension can use **rqe_dos_request** to obtain the current PSP. This enables the VM86 extension to track which resources are allocated to which DOS program.

To ensure that the PSP address obtained is the PSP of the current DOS program, rather than that of the RMX TSR program, set the `tsr_flag` parameter to 1 in the **rqe_dos_request** call.

Extension System Call Restrictions

The extensions called by the VM86 Dispatcher can use only system calls in the BIOS and Nucleus subsystems of the iRMX OS. Extensions run in the context of the VM86 Dispatcher Job, and can only make the same system calls as the Dispatcher Job.

Extension Installation Examples

This section discusses three code segments which illustrate:

- Installing an extension from the iRMX OS
- Two ways of initiating an extension from DOS

Installing an Extension from the iRMX Operating System

An iRMX program *\rmx386\demo\c\vm86ext\rmxext.c* illustrates how to install an extension. It also gives example code for both the VM86 Extension entry procedure and the deletion handler.

In the example, `main()` creates a mailbox, prints an installation message, and installs the extension for interrupt level 0B9H (185 decimal). It then waits to receive a message at the mailbox. At this point, `main()`waits until the DOS part of the example issues INT 0B9H.

When the DOS part issues INT 0B9H, `main()` calls the extension procedure. The `entry_procedure` extension procedure sends a message to the mailbox created in `main()`, where `main()` is waiting. `Main()` receives the message, prints it out, and deletes in order, the segment described by `segtoken`, the mailbox, and the extension.

The example was compiled and bound using Intel's iC-386 compiler and BND386 binder. The mailbox token, `mbxtoken`, is an iRMX object.

Initiating an Extension from DOS

The DOS part of the example uses two programs. The C program is in *\rmx386\demo\c\vm86ext\dosext.c*, and the assembly language program is in *\rmx386\demo\c\vm86ext\dosext.asm*. These two programs issue INT 0B9H (185 decimal). Both examples have the same functionality.

The DOS application C program illustrates one way the previously installed extension can be initiated. The example was compiled and linked using the Microsoft Version 7.0 C compiler.

The DOS application assembly program illustrates one way the previously installed extension can be initiated. The example was created using the Microsoft Version 5.1 assembler.

❑❑❑

Making DOS and ROM BIOS System Calls 4

This chapter describes how to make DOS and ROM BIOS system calls from iRMX application programs.

Making DOS and ROM BIOS Calls from an iRMX Application

The **rqe_dos_request** system call enables the iRMX OS to make ROM BIOS and DOS requests in much the same way as the RTE system calls allow a DOS application program to make iRMX system calls.

Using **rqe_dos_request**, a DOS application program can be ported to an iRMX environment to take advantage of the Protected Mode features without changing all DOS and ROM BIOS calls to iRMX system calls.

The application program can also use the **rqe_dos_request** system call to access a DOS device driver which may be running in VM86 Mode. However, the application program must not use a DOS system call that conflicts with the file server.

The DOS data structure represents the microprocessor registers, and the **rqe_dos_request** system call passes a pointer to this structure.

See also: **rqe_dos_request** and DOS data structure, *System Call Reference*

To make DOS/ROM BIOS calls, the application program must set the appropriate register values in the structure pointed to by `register_ptr`, set the `int_num` parameter with the required DOS interrupt level, and set the `xfer_data` byte, the source and destination transfer pairs, pointers, and counts based on the data being transferred. The **rqe_dos_request** system call can then be invoked and the required DOS system call will be made. The WORD pointed to by the `status_ptr` parameter contains the condition code generated by the **rqe_dos_request**. If the call was successful, the structure pointed to by the `register_ptr` parameter reflects the register values returned by the DOS system call.

Many DOS system calls pass data. This can be done by passing the segment base and offset of the source or destination address, where data is located in one or more register pairs. Other registers sometimes specify the length of data located at the address specified by the register pair.

The DOS system call **Get Redirection List Entry**, Interrupt 21H, Function 5FH, Subfunction 02H, uses the DS:SI register pair to point to a 16-byte (maximum) character string containing a device name in the redirection list. The ES:DI register pair points to a 128-byte (maximum) character string containing the network name of the device. The data transfers from the system call to the calling application program.

To make a similar call using the **rqe_dos_request** system call, you use four separate sets of structure elements to control the data transfer. These structure elements indicate the appropriate registers, but are ignored if the `xfer_data` structure element is set to 0.

See also: **rqe_dos_request**, *System Call Reference*

Example: Get Free Disk Space

The **Get Free Disk Space** DOS system call transfers data only in microprocessor registers. To make the DOS system call **Get Free Disk Space** (of a DOS drive) Interrupt 21H, Function 36H, the iRMX application would:

1. Set `int_num` (DOS system call interrupt number) to 21H.
2. Set `reg_ah` (Interrupt 21H subfunction code) to 36H.
3. Set `reg_dl` (Drive code) to the required drive code level where 1 = drive A, 2 = Drive B, etc.
4. Set `xfer_data` to 0, as no data is transferred with this system call except directly using the microprocessor registers.
5. Set `status_ptr` to point to a WORD variable, which the **rqe_dos_request** system call sets with a condition code before returning to the iRMX application program.
6. Make the **rqe_dos_request** system call.

See also: **rqe_dos_request**, *System Call Reference*

To determine the result of the requested DOS system call, the application program would then:

7. Determine the value of the WORD variable pointed to by `status_ptr`. If the **rqe_dos_request** call did not succeed (condition code not equal to E_OK), the application program terminates with an appropriate error message.
8. If the condition code returned was 0, the application program could proceed.
9. The values stored in `reg_al` and `reg_ah`, on return from the call, hold these values:

`reg_ah` = FFH `reg_al` = FFH	DOS determined that the drive code specified by `reg_dl` was not valid.

or

`reg_ah`<>FFH `reg_al`<>FFH	`reg_ah`, `reg_al` specifies the sectors per cluster of the specified drive.

10. If the drive code was valid, the other register values are set as:

`reg_bh`, `reg_bl`	Specifies the number of available clusters on the specified drive.
`reg_ch`, `reg_cl`	Specifies the number of bytes per sector on the specified drive.
`reg_dh`, `reg_dl`	Specifies the number of clusters (used or available) on the specified drive.

Get Redirection List Entry Example

The example *\demo\c\vm86ext\dosdevs.c* uses the DOS system call **Get Redirection List Entry**. The first part shows to set registers in the DOS data structure prior to making the **rqe_dos_request** system call. The second part shows how to make the **rqe_dos_request** system call. The example was compiled and bound using Intel's iC-386 compiler and BND386 Binder.

Setting the DOS Data Structure

The DOS system call **Get Redirection List Entry**, Interrupt 21H, Function 5FH, Subfunction 02H, uses the DS:SI register pair to point to an ASCIIZ (null-terminated) character string defining the local device name found in the list, and uses the ES:DI register pair to point to the ASCIIZ character string defining the network name for that local device.

For this example, though no source data is transferred, two character strings are created by the DOS system call destination data parameter, which are pointed to by the two register pairs.

Set the DOS data structure as follows:

1. Set `int_num` (DOS system call interrupt number) to 21H.
2. Set `tsr_flag` to 0.
3. Set `reg_ah` (Interrupt 21H function code) to 5FH.
4. Set `reg_al` (Function 5FH subfunction code) to 02H.
5. Set `reg_bx` to the required redirection list index.
6. Set `xfer_data` to 0FFH since data is transferred with this system call.
7. Set `src1_xfer_pair` and `src2_xfer_pair` to 0 since no source data transfer is required.
8. Set up the destination data control parameters for the local device name as follows:

`dest_p_1`	&local_device_name (local_device_name is a character array)
`dest1_xfer_pair`	4 (to specify the DS:SI register pair)
`dest_count_1`	16 (to specify a maximum size of 16 characters)

9. Set up the destination data control parameters for the network name as follows:

`dest_ptr_2`	&network_name (network_name is a character array)
`dest2_xfer_pair`	8 (to specify the ES:DI register pair)
`dest_count_2`	128 (to specify a maximum size of 128 characters)

10. Set `status_ptr` to point to a WORD variable which the **rqe_dos_request** system call will set with a condition code before returning to the iRMX application.

11. Make the **rqe_dos_request** system call.

    The **rqe_dos_request** system call returns a value in the WORD variable pointed to by `status_ptr`. The meanings of the values are:

Value	**Meaning**
Not 0	The call encountered an error, as specified by the error code. The iRMX application needs to evaluate the error to see if a retry is possible.
0	Since no errors were encountered by the iRMX OS, the application can proceed.

For certain DOS system calls, including this example, the carry flag is set to one of two values.

Value	**Meaning**
1	The DOS system call failed. If the call failed, `reg_al` will contain the DOS error code.
0	The DOS system call was successful.

In this example, if the DOS system call is successful, these parameters return to the caller from the DOS system call:

`reg_bh`	Device status flag
`reg_bl`	Device type
`reg_cx`	Stored parameter value

❑❑❑

General Information 5

This chapter describes various information related to DOSRMX. The information includes such areas as programming techniques, use of iRMX objects, and network concepts. This information does not apply to the iRMX III or iRMX for PCs OSs.

Interrupt Virtualization and Determinism

DOSRMX has two interrupts modes in which DOS can operate. You can configure the mode by changing the appropriate setting in the *\rmx386\config\rmx.ini* file. These modes are Interrupt Virtualization Enabled (VIE=0FFH) and Interrupt Virtualization Disabled (VIE=00H). The default is Interrupt Virtualization Disabled.

With Interrupt Virtualization disabled, DOS executes all real mode instructions supported by the microprocessor, including ENABLE and DISABLE INTERRUPTs. This affects system performance in interrupt latency and determinism by enabling DOS and ROM BIOS-based disk I/O to operate at near optimum DOS performance levels. In this mode, the iRMX OS has little interaction with DOS.

With Interrupt Virtualization enabled, the iRMX OS traps all DOS access of privileged instructions (CLI, STI, INT, POPF, etc) as well as attempts to access the Programmable Interrupt Clock and Programmable Interrupt Timer. The iRMX OS can virtualize interrupts with respect to DOS, while keeping interrupts disabled for the iRMX OS for as short a time as possible. This increases interrupt response time and decreases interrupt latency for iRMX-owned interrupt levels, such as non-DOS levels. However, this frequent intervention by the iRMX OS into DOS operations also affects DOS and ROM BIOS-based I/O performance. Small transfers slow down dramatically while larger transfers (4 Kbytes or larger) experience relatively little degradation.

Based on the needs of the application, DOSRMX can be optimized for either higher performance DOS/ROM BIOS-based I/O with less than ideal determinism or for solid determinism with less than ideal DOS/ROM BIOS-based I/O performance. To get both solid determinism and high performance I/O, first add an iRMX-owned disk controller such as the Adaptec 1542, then use the PCI loadable driver provided in the product, and finally set VIE=0FFH in the *rmx.ini* file.

See also: *rmx.ini* file, *System Configuration and Administration*

Real-time Fence

The real time fence is set at priority level 127. All active DOS-owned interrupts are temporarily masked when tasks are running at or above (numerically lower than) priority level 127. The real-time fence here is different from the real-time fence used with round robin scheduling. This preserves the real time aspect on the iRMX OS side of an DOSRMX application system. If you make an **rqe_dos_request** call from an iRMX task running at a priority above or equal to this real-time fence, you will receive an E_TIME condition.

See also: Real-time fence, *System Configuration and Administration*

iRMX-NET Access From a DOS Server

You can access a DOS server indirectly from the DOSRMX side using the EDOS File Driver. For example:

From the DOS side of DOSRMX, do this:

```
net use r: \\<dos-server>
dir r:
```

Then, from the iRMX side of DOSRMX, do this:

```
ad r_dos as :r: e
dir :r:
```

See also: Using iRMX-NET in a DOS Environment, *System Configuration and Administration*

If you try to directly access a DOS server from the iRMX OS, such as with the **attachdevice** command, you may encounter a General Protection fault or similar failure.

❑❑❑

DOSRMX Default Configuration A

This appendix lists the pre-configured options in the software definition file, used to generate the DOSRMX boot image. If you ported an existing application to DOSRMX, you may need to alter it to run within the pre-configured software. If your application is incompatible with this configuration, use the Interactive Configuration Utility to change it.

See also: Definition files, *ICU User's Guide and Quick Reference*, for a listing of the definition files you can customize.

Tables of pre-configured options are provided for these system requirements and sub-systems:

- Sub-Systems
- Memory
- Human Interface
- Application Loader
- Extended I/O System
- Basic I/O System
- Device Drivers
- System Debug Monitor
- Nucleus
- Nucleus Communication Service
- VM86 Dispatcher Reserved Interrupts

Sub-System Configuration

Table A-1. Sub-Systems Options

Sub-Systems	Default
Universal Development Interface	Yes
Shared C Libraries	No
Human Interface	Yes
Application Loader	Yes
Network Access	No
Extended I/O System	Yes
Basic I/O System	Yes
System Debug Monitor	Yes
System Debugger	No
OS Extension	Yes

Memory Configuration

Table A-2. Memory Options

Memory for System	Default
Start Address	110000H
End Address	1FFFFFH
Memory for Free Space	**Default**
Start Address	0200000H
End Address	0FFFFFFFFH

Human Interface Configuration

Table A-3. Human Interface Options

HI Jobs	**Default**
Jobs Minimum Memory	0H
Jobs Maximum Memory	0FFFFFFFFH
Numeric Processor Extension Used	Yes
Prefixes	**Default**
Prefix :	:PROG:
Prefix :	:UTILS:
Prefix :	:UTIL286:
Prefix :	:SYSTEM:
Prefix :	:LANG:
Prefix :	:$:
HI Logical Names	**Default**
Name = WORK	:SD:WORK
Name = UTILS	:SD:UTIL386
Name = UTIL286	:SD:UTIL286
Name = LANG	:SD:LANG286
Name = RMX	:SD:RMX386
Name = INCLUDE	:SD:INTEL/INCLUDE

Application Loader Configuration

Table A-4. Application Loader Options

Application Loader	**Default**
All System Calls	Yes
Default Memory Pool Size	0500H
Read Buffer Size	01000H

Extended I/O System Configuration

Table A-5. EIOS Options

EIOS	Default
Retries on Physical Attachdevice	0H
Default IO Job Directory Size	200
Automatic Boot Device Recognition	**Default**
Default System Device Physical Name	C_RMX
Logical Names	**Default (Device Name, File Driver, Owner's ID)**
Logical Name = BB	BB, PHYSICAL, 0H
Logical Name = Stream	STREAM, STREAM, 0H

Basic I/O System Configuration

Table A-6. BIOS Options

BIOS	Default
Attach Device Task Priority	129
Timing Facilities Required	Yes
Timer Task Priority	129
Connection Job Delete Priority	130
Ability to Create Existing Files	Yes
System Manager ID	Yes
Common Update Timeout	1000
Terminal Support Code	Yes
Control-Sequence Translation	Yes
Terminal OSC Controls	Yes
Tape Support	No
BIOS Pool Minimum	0800H
BIOS Pool Maximum	0FFFFFH
Global Clock	ATRT
Global Clock Name*	

* The Global Clock Name has a blank string as a default.

Device Drivers Configuration

Table A-7. Device Drivers Options

Driver	Default
AT Serial Driver	
DUIB Name	COM1
Interrupt Level	048H
Base Port Address	03F8H
Reset Character	0H
Interrupt Character	0H
DUIB Name	COM2
Interrupt Level	038H
Base Port Address	02F8H
Reset Character	0H
Interrupt Character	0H
ROM-BIOS Based Hard Disk Driver	
DUIB Name	C_RMX and D_RMX (first iRMX partition) C_RMX0 and D_RMX0 (whole physical drive) C_RMX1 through C_RMX4 and D_RMX1 through D_RMX4
Base I/O Port Address	01F0H
Control/Status Port Address	03F6H
ROM-BIOS Based Diskette Driver	
DUIB Name	A and B (5.25 inch format, 360 Kbyte) AH and BH (5.25 inch format, 1.2 Mbyte) AM and BM (3.5 inch format, 720 Kbyte) AMH and BMH (3.5 inch, 1.44 Mbyte) AMO and BMO (3.5 inch, 2.88 Mbyte)
Interrupt Timeout	01770H
EDOS File Driver	
DUIB Name	A_DOS, ... ,Z_DOS
DOS File Driver	
DUIB Name	C_DOS, ... ,Z_DOS

System Debug Monitor Configuration

Table A-8. System Debug Monitor Options

System Debug Monitor	Default
Console Port	System Console Primary

Nucleus Configuration

Table A-9. Nucleus Options

Nucleus	Default
Number of GDT Entries	8000
Number of IDT Entries	256
Parameter Validation	Yes
Root Object Directory Size	200
Default Exception Handler	SDB
NMI Exception Handler	IGNORE
NMI Enable Byte	4
Exception Handler for Stack Exception	SDB
Name of Ex Handler Object Module	
Exception Mode	NEVER
Low GDT/LDT Slot Excluded from FSM	0
High GDT/LDT Slot Excluded from FSM	0
Round Robin Priority Threshold	140
Round Robin Time Quota	5
Report Initialization Errors	YES
Maximum Data Chain Elements	0
Nucleus Communication Service	YES

Nucleus Communication Service Configuration

Table A-10. Nucleus Communication Service Options

Nucleus Communication Service	Default
Message Task Priority	128
Deletion Task Priority	128
Default Number of Port Transactions	16
Default Host ID	0
Validate Buffer Parameters	Yes
Max. No. of Simultaneous Transactions	080H
Max. No. of Simultaneous Messages	0100H
Receive Fragment Failsafe Timeout	0400H
Number of Trace Messages	255

VM86 Dispatcher Reserved Interrupts Configuration

Table A-11. DOS Extender Reserved Interrupts

DOS Extender Reserved Interrupts	Default
Master Level 0	Clock
Master Level 2	Slave PIC

❑❑❑

Index