Real Time Disk Operating System

(RDOS)

Reference Manual

093-000075-08

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000075

Printed in the United States of America
Revision 08, March 1979

Contents

Chapter 1 - Introduction

Chapter 2 - Files and Directories

Chapter 2 - Files and Directories (Continued)

Chapter 3 - Single Task Programming

Chapter 3 - Single Task Programming (Continued)

Chapter 4 - Extended User Address Space: Swaps, Chains, User Overlays, Window Mapping, and Extended Memory I/O

Chapter 5 - Multitask Programming

Chapter 5 - Multitask Programming (Continued)

Chapter 6 - Foreground-Background Programming

Appendix A - RDOS Command and Error Summary

Appendix B - User Parameters

Appendix C - Interrevision Changes

Appendix D - Real-Time Programming Examples

Appendix E - Overlay Directory Structure

Appendix F - Exceptional System Status

Appendix G - Bootstrapping RDOS from Disk

Appendix H - Hollerith-ASCII Conversion Table

Appendix I - ASCII Character Set

Appendix J - Advanced Multitask Programming

Appendix J - Advanced Multitask Programming (Continued)

Illustrations

Figure Caption

Tables

Table **Caption**

Chapter 1
Introduction

Data General's Real Time Disk Operating System (RDOS) combines the advantages of a disk operating system with the speed of a memory-resident system. RDOS is real-time oriented; it can allocate program control to many tasks within separate foreground and background programs. RDOS offers maximum system efficiency, economically, to a wide variety of installations.

Some major features of RDOS are:

- Disk and memory-residence
- Support for real-time FORTRAN IV, FORTRAN 5, ALGOL, Extended and Business BASIC, and other advanced languages
- Support for BATCH processing
- Flexible file structure: Disk partitioning and sharing of user files; buffered and unbuffered I/O; multiple user overlays
- Modular multitask levels of task priority
- 256 software levels of task priority
- Hardware mapping support for foreground/background programming: Protection and management of each program; access to mapped extended memory; checkpointing of background programs
- Spooling (disk buffering) of output to slow peripherals
- Dual processor-shared disk support
- Multiprocessor support
- Tuning for improved performance

Consult the *Introduction to RDOS* for an introduction to these and other basic concepts in RDOS.

To use RDOS you need a suitable Data General computer, a console teletypewriter or CRT display, and a disk. Larger versions of RDOS can support a real-time clock, power fail-auto restart, up to 16 megabytes of fixed-head disk storage and more than 1,500 megabytes of moving-head disk storage. RDOS can support 16 mag tape and 16 cassette units, multiple line printers, terminals, plotters, reader/punches, multiplexors, and CPUs. Mapped RDOS features hardware memory protection, and can support up to 256K bytes of memory (NOVA) or 512K bytes (ECLIPSE).

Generating an RDOS System

Each system installation is unique; it must perform diverse tasks with one of many possible hardware combinations. You can tailor RDOS for your own environment with the system generation procedure (SYSGEN).

SYSGEN, the builder of tailored operating systems, is an executable system program which can operate in any installation. A standardized starter (bootstrap) system was delivered with your RDOS system; this starter system and SYSGEN enable you to generate one or more configured systems. If you know your future requirements, you can generate other RDOS systems at this time to fulfill them. A separate manual, *How to Load and Generate Your RDOS System*, describes all SYSGEN procedures.

You bootstrap a generated system into execution via BOOT, the RDOS bootstrap program. Appendix G contains a convenient summary of RDOS disk bootstrap procedures.

Communicating With RDOS

You can communicate with RDOS, and make it work for you, in four ways:

- via system and task calls in an assembly-language program; or
- through Command Line Interpreter commands; or
- through the Batch monitor; or
- indirectly, through a higher-level language.

You write system and task calls as instructions in a program, and use the CLI as a dynamic console interface to RDOS. System and task calls activate logic within system or task processing modules. Only those task-processing modules which your program needs become part of it.

The Command Line Interpreter (CLI) is a system utility program that accepts command lines from the console and translates them into commands to RDOS. Thus, the CLI is an interface between your console and the system.

RDOS restores the CLI to memory whenever RDOS is idle--after initialization, after a disk bootstrap, after a console break, after the execution of a program, etc. The CLI indicates that it is in control by outputting a ready message prompt, "R", and a carriage return.

You activate the CLI by entering a CLI command via the console. You can interrupt the CLI's action by pressing the keys CTRL and A, CTRL and C, or CTRL and F. We'll describe these keyboard interrupts near the end of Chapter 3.

CLI commands allow you to load programs, invoke other utility programs, and activate the BATCH monitor. BATCH executes jobs serially, without operator intervention, using job control commands in the job stream.

Advanced Data General compilers, and the BASIC interpreter, allow you to write programs in languages like ALGOL, FORTRAN, and BASIC.

Program Development

Along with the CLI, you received a number of utility programs from Data General. Each program is described in a separate manual which you also received. The utilities help you write code and develop it into useful, executable programs. During system generation, (manual: *How to Load and Generate Your RDOS System*) you transferred the utility programs to disk; each utility then became accessible by a CLI command.

Your first step in program development is to write a source program which does useful work for your computer application. You can do this in a higher-level language like ALGOL or FORTRAN, or in assembly language, via one of the text editor utilities: the Text Editor (EDIT command), Multiuser Text Editor (MEDIT command), or Supereditor (NSPEED or SPEED command). Your next step depends on whether you have used a higher-level language like FORTRAN, or assembly language. This manual will be most useful to assembly-language requirements.

Higher-Level Languages

If you have written your program in FORTRAN, ALGOL, or another higher-level language, you will compile and assemble it with the appropriate compiler utility (FORT command for FORTRAN IV, ALGOL command for ALGOL, etc.) You will then process it to produce an executable program file via the Relocatable Loader utility (RLDR command). If you have written the program using the BASIC interpreter, you can execute the program directly in BASIC. You check and correct any errors using the appropriate manual for your language, and use the CLI to access, maintain, and protect your files and devices.

Assembly Language

If you have written your source program in assembly language via the Text Editor or Supereditor Utility, you must assemble it into a relocatable binary file (ASM or MAC commands). After you assemble your source program into a binary, you'll use another utility to process the binary into an executable program file (called a save file). This utility is the Relocatable Loader (RLDR command). The first time you load a program, it will usually need debugging; you therefore load with it a symbolic debugger utility. You can then try to execute the program, and if it doesn't run properly, debug it (DEB command). After any step, you can use other CLI commands to maintain, protect, and examine the file.

Main Memory Considerations

Your computer arrived with a given amount of memory. The amount of this memory which is available for your programs will necessariiy be a percentage of this figure, as determined by the requirements of the RDOS system you generated. The peripherals and software structures you specified at SYSGEN each require some memory, as described in *How to Load and Generate Your RDOS System.* After deducting the RDOS total from your maximum figure, you must also consider the space (aside from your own code) that the program will actually require.

When you load a program, RLDR builds certain required tables, modules, directories, and the required Task Scheduler into it. The code for each task call you use is taken from your system library and loaded into the program (*system* calls are executed in RDOS space, hence require little user space). These all require user memory space when you execute the program - and you may want to conserve space by coding certain segments of the program as *overlays*. Overlays are called into memory one-by-one, as the program needs them, and otherwise stay on disk. You define overlays within a program in the RLDR command line. Another way to extend effective user address space is to have an executing program swap itself to disk, call a whole new program into memory, and return to memory when the new program has executed. This is called swapping, and it has a variation called chaining. Overlays, swaps, and chains are described in Chapter 4. This chapter also explains extended memory - which you can access if you have a mapped machine.

Foreground/Background Programming

According to your installation, you may want to run two logically distinct programs concurrently. RDOS allows you to divide memory into two areas, called foreground and background, and run a program simultaneously in each. When you bootstrap your system, RDOS starts up in the background; all programs you execute will run in the background until you command RDOS to execute one in the foreground. When you run two grounds, they share such system resouces as CPU time and I/O devices. The foreground program has priority, unless you specify equal priority. The foreground and background programs can communicate with one another via system calls or commonly-known disk files. For more on this, see Chapter 6.

If your system does not have a hardware mapping device, you have an unmapped system which runs under unmapped RDOS; if you want to run a program in the foreground, you must manually assign memory to it. You do this in the RLDR command line by specifying two starting addresses for the foregound program. These are the start of page zero relocatable memory (called ZREL) and the start of normal relocatable memory (called NREL). After you load the program, and execute it in the foreground, the addresses you specified will separate the background and foreground. In an unmapped system, you can directly address up to 32K words of user address space, excluding RDOS system space.

Certain system calls, features, and CLI commands apply only to mapped systems. We will note these exceptions in the text; if you see no reference to mapping or the MAP in a command or feature, assume that it applies to both mapped and unmapped systems.

Mapped Features

If you have a hardware map, then you have a mapped system which runs under mapped RDOS. In mapped RDOS, background and foreground programs can operate autonomously, either alone or via a CLI; using mapped address space, both programs can share all memory not used by RDOS. Naturally, this depends on the amount of memory your machine has and the size of your RDOS system, as determined by the features selected during SYSGEN. Tools for accessing extended memory include virtual overlays and window mapping (Chapter 4). Any Data General computer with mapping hardware can support mapped RDOS.

In a mapped system, you specify addresses as you do in an unmapped system, but the system can remap them in pages of 1,024 words. Addresses in mapped systems are called logical addresses, instead of physical addresses.

When you run two programs, the system maps each program separately; each program is aware of its own address space only, and cannot reference locations outside it. The system allots memory to each program according to its highest address.

The system assigns each program a complete logical address space from page zero through its highest address NMAX, in 1,024-word pages.

When RDOS starts up, it assigns all memory to the background; you reserve memory for the foreground with the CLI command SMEM, and execute a program in the foreground with the EXFG command.

Aside from hardware separation of foreground and background, the mapped system protects itself in three ways: it guards system devices, prevents infinite address defers, and protects data channel operations.

Initially, no one can access any device directly (including the MAP and CPU) on a machine-language level. If anyone tries to reference a device on a machine level, without having been enabled to do so, the system will refuse the request, print a "trap" message, create a break save file, and return to a higher-level program - usually the CLI. The system will do the same thing if it encounters more than 16 levels of indirect address - trap, create the break file, and return. We describe traps in Appendix F.

Users can gain direct access to any system device- and avoid the map's safeguards- by using the system call .DEBL (Chapter 3).

The map also monitors the data channel, and allows user devices to access it through the system call .STMAP (Chapter 7).

RDOS Organization

The RDOS executive is the main framework of the operating system and must be memory-resident before any processing can occur. This resident portion of RDOS processes system calls and interrupts, and manages RDOS buffers. Other modules of the system reside in system overlays. These are brought into memory from disk storage as required to perform specific functions like initializing the system, opening, closing, renaming or deleting files, and spooling control.

In an unmapped system, the RDOS executive resides at the top and bottom of memory. Locations 0 through 15_8 contain program and interrupt entry points into the top area of RDOS. In a mapped system, resident RDOS begins at location 0 and extends to the highest address required; it is invisible to your programs. Above resident RDOS in all systems (at the very top of memory in unmapped systems) is a series of system buffers. The system buffers handle buffered I/O transfers, and hold system overlays and directories from disk.

The portion of page zero memory available for your programs begins at location 16_8 (labelled USP), skips to locations 20_8 though 37_8; then extends from 50_8 through address 377_8. In an unmapped system, these are physical addresses; in a mapped system, they are logical addresses. NREL memory is allocated in much the same way for both mapped and unmapped systems. In a mapped system, ZREL and NREL addresses are logical; in an unmapped system, they are absolute, but your programs won't care about this distinction.

Above program ZREL, the Relocatable Loader (RLDR) builds a User Status Table (called UST) for your program. This table starts at address 400_8 in an unmapped background, and logical address 400_8 in both the mapped foreground and mapped background areas. The UST describes, among other things, your program's length, number of tasks required, and number of I/O channels needed.

Above the UST RDOS reserves an area for a pool of Task Control Blocks (TCBs). RDOS uses TCBs to store task state information, such as the state of the accumulators and carry. If you have defined overlays in your program via RLDR, an overlay directory sits above the TCBs. Above the overlay directory (if any) is NREL memory, which holds the rest of your program. RLDR reserves a node (vacant space) in your program for each overlay segment you defined; overlays from each group will occupy this node one-by-one.

Above your program (background or foreground), still in NREL memory, are the task-processing modules and Task Scheduler which it requires to run. RLDR searched the system library for these, and placed them on disk with your program. Generally, during execution, they are highest in NREL memory.

Figure 1-1, below, is a simplified illustration of unmapped and mapped memory. Each system is running foreground and background programs; each program has one overlay node.

System Library and Source Files

Your system library, named SYS.LB, contains task-processing modules, task schedulers, and other useful routines for your programs.

Other files supplied with your system contain definitions for system features and for system and user parameters. Depending on the programs you write, you may want to include some or all of these files in the Macroassembler's permanent symbol file (MAC.PS), as described in the *Macroassembler User's Manual.* Naturally, you can LIST, PRINT or type any of these files. Table 1-1 lists the names of the most common of these files.

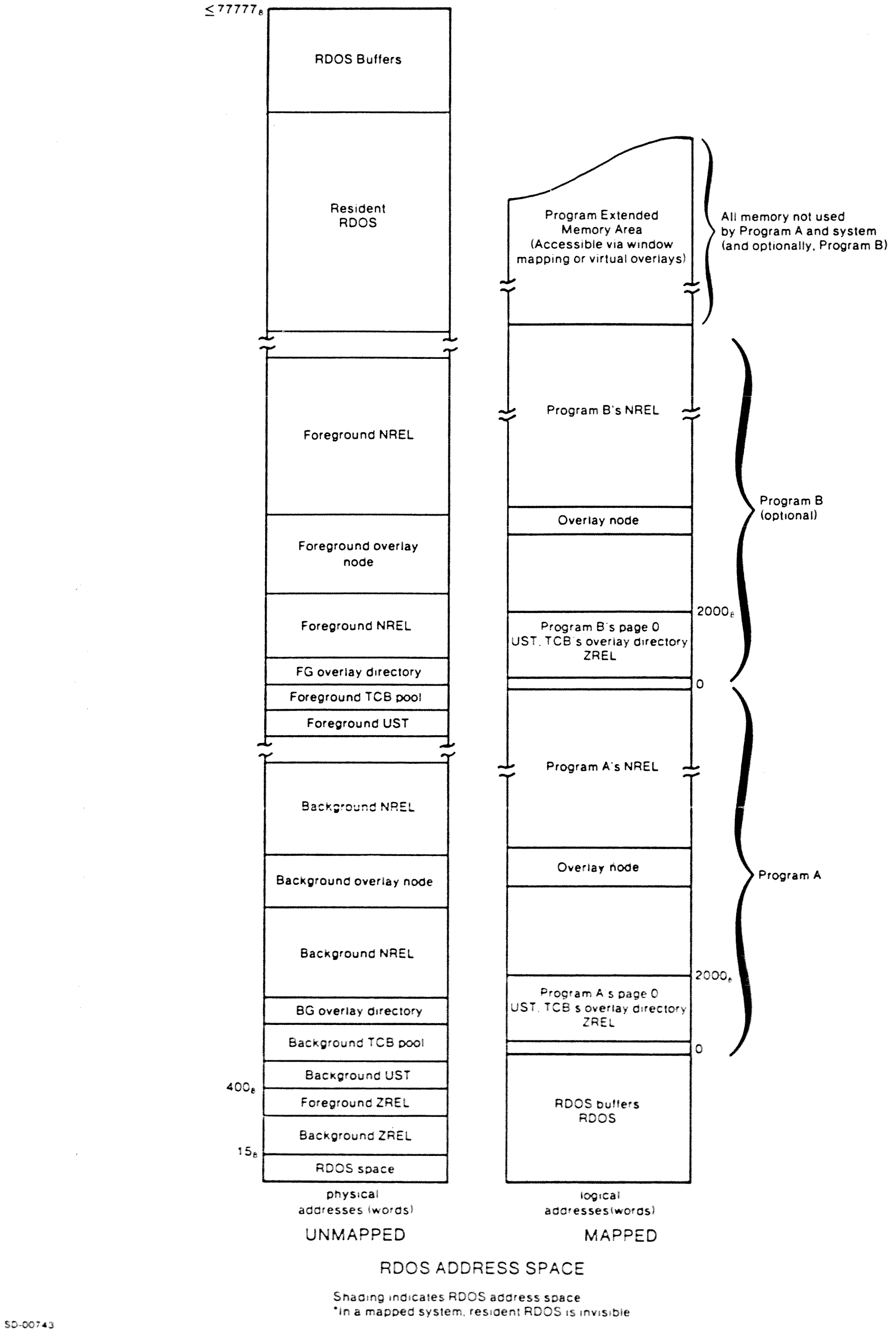


Figure 1-1. RDOS Address Space

Table 1-1. System File Names

All Data General computers:

You can use these files, in addition to the machine-specific files described below, on all DG machines.

User parameter file: PARU.SR contains mnemonics for all system constants and errors; you will probably use these extensively for assembly-language programming. Appendix B contains a listing for PARU.SR.

System parameter file: PARS.SR contains internal RDOS constants and some macros for system-level tables, such as device control blocks and certain buffers.

NOVA Basic Instruction Definition: NBID.SR provides the basic instruction set for all DG machines.

Operating System Instruction Definition: OSID.SR.

Multiply-divide Instructions: provided with your language.

RDOS literal macros: LITMACS.SR. RDOS uses these macros; you can also use them for your own programs.

Floating-point instructions:

NFPID.SR	For machines with hardware floating point
FPID.SR	For machines with software floating point (floating point interpreter package)

For manipulating AC3: See *Status on Return from System Calls,* Chapter 3, for a description of these modules.

For ALM multiplexors: ALMSPD.SR. You can edit this file to reflect your line configuration. It is further described under *Multiplexors* in Chapter 2.

NOVA 3 Computers:

Stack Instruction Definition: NSID.SR

Specific RDOS system instructions

Unmapped NOVA3s:	RDOS.SR
Mapped NOVA3s:	NRDOS.SR

Other NOVA Computers:

Specific RDOS Instructions

Unmapped:	RDOS.SR
Mapped:	MRDOS.SR

ECLIPSE Computers:

NOVA Extended Instruction Definitions: NEID.SR
Commercial ECLIPSE Instructions: NCID.SR
Hardware floating-point instructions: NFPID.SR
Specific RDOS Instructions

Unmapped:	BRDOS.SR
Mapped:	128K:ARDOS.SR
	256K:ZRDOS.SR

End of Chapter

Chapter 2
Files and Directories

This chapter defines the different RDOS media for files - generally disk, and mag tape- and explains how to use each medium. It ends with a description of multiplexors. The section on disk files describes the mechanisms which you can use to organize and speed up access to your disk files, and it outlines the file structure which RDOS imposes on every disk it uses. These mechanisms include directories - called partitions and subdirectories- which contain groups of files, and link entries. Link entries allow users in different directories to use a single file. For a practical introduction to files, subdirectories, and partitions, run through the console session in Chapter 2 of *Learning to Use Your RDOS/DOS system.*

Definition of a File

A *file* is any collection of information or one of several devices for receiving or sending the information. Typical examples of both file types are:

- Source file
- Relocatable binary file
- Executable program file (save file)
- Listing file
- Teletypewriter or CRT keyboard
- Teletypewriter printer or CRT screen
- Paper tape reader or line printer
- Cassette or magnetic tape file

Each of the first four file types has certain characteristics, and represents a step in program development. You write a *source file* with a text editor, and input it to an assembler which produces as output a *relocatable binary file.* You process the relocatable binary file with the loader; this places the file on disk with absolute location data as a *save* file. The save file is the executable program version of the file. Each save file is a *core-image* file; it is stored on disk word-for-word as it will be loaded into memory and executed. You can create a listing file to store and/or output the result of any of these steps. RDOS executes the assembly and loading step via CLI commands.

Unless you specify another file, the *keyboard* and *printer or screen* are the default input and output files for most system operations. The paper tape reader is another type of input file; the line printer, another type of output file.

Cassette or magnetic tape files are discussed briefly below, and extensively later in this chapter.

File Overview

You access all devices and disk files by file name; you access all cassette and magnetic tape files by device name and file number.

You must open a file (i.e., associate it with an RDOS channel via an OPEN system call) before you can access it. The CLI file I/O commands do this automatically, but when you program in RDOS, your program must open any files it needs. You can open a disk file and allow several concurrent users to access and modify the file's contents; or you can open it exclusively, permitting only one user to modify the file but permitting other users to read the file; or you can open it for reading only by several users.

Reserved Device Names

I/O devices have special names which often begin with the character $. Within the limits of the device, you can use each device name exactly as you would a disk file name in a command. You enter each device name as shown in Table 2-1, below.

Table 2-1 Reserved Device Names

Device Name	Device
	Asynchronous Line Multiplexor (see QTY for device name).
$CDR	Punched card reader; mark sense card reader.
CTn	Data General cassette unit n, first controller (n is in the range 0-7)
DK0	Data General model 6001-6008 fixed-head disk, first controller.
DPn	Data General moving-head disk pack, first controller, unit n is 0, 1, 2, or 3.
DPnF	Top loader (dual-platter Disk Subsystem). For the first controller, unit n is number 0, 1, 2, or 3. This unit has two disks. The top (removable) disk is DPn, the fixed disk is DPnF. This controller also supports diskette drives.
DSn	Data General Model 6063/6064 fixed head disk. The 6063 is single-density, the 6064 double density. n is 0, 1, 2, or 3.
DZn	6060 series disk unit, first controller. n is 0,1,2,or 3. The 6060 uses single-density disks, 6061 uses double-density disks.
$DPI	Input dual processor link (see Chapter 8).

Device Name	Device
$DPO	Output dual processor link (see Chapter 8).
$LPT	80- or 132-column line printer.
MCAR	Multiprocessor communications adapter receiver.
MCAT	Multiprocessor communications adapter transmitter.
MTn	First controller, 7- or 9-track magnetic tape transport n, (n is in range 0-7).
$PLT	Incremental plotter.
$PTP	High-speed paper tape punch.
$PTR	High-speed paper tape reader.
QTY	Asynchronous Line Multiplexor (ALM), asynchronous data communications multiplexor (QTY), or Universal Line Multiplexor (ULM).
$TTI	Teletypewriter or display terminal keyboard*.
$TTO	Teletypewriter printer or CRT display.
$TTP	Teletypewriter punch.
$TTR	Teletypewriter reader.

*For most devices, RDOS supplies an end of file mark. On $TTI and QTY input, however, you must indicate an end-of-file by pressing the CTRL and Z keys (CTRL Z).

Aside from the ALM and QTY, we have written the device drivers reentrantly to allow RDOS to support pairs of devices. Use the following names to address second device controllers on your system.

DK1 Second Data General fixed-head disk.

DPn Second Moving-head disk pack controller (n is 4, 5, 6, or 7).

DPnF Second top loader (model 6045 or 4234A) controller; n is 4, 5, 6, or 7. The removable disk is DPn; the fixed disk is DPnF.

DSn Second 6063/6064 fixed-head disk controller. n is 4, 5, 6, or 7.

DZn 6060-series unit, second controller. n is 4, 5, 6, or 7.

CTn Second cassette controller, n is 10-17_8.

MTn Second mag tape controller, n is 10-17_8.

For other devices, append a "1" to the primary device name; e.g., $LPT1, $PTP1, $CDR1, and so on.

Disk File Names

A disk file name is a string of up to 10 ASCII characters, including upper and lower case letters, numbers, and $. (By default, RDOS converts lower case letters to uppercase.) The string is packed left to right, and terminated by a carriage return, form feed, space or null. You can use any number of characters in a file name, but the system recognizes only the first 10. Moreover, you can use $ whenever you want in a disk file name; but generally, you should avoid the reserved device name combinations.

You can append an extension to any disk file name. An extension is a period and one or two alphanumeric characters, which may include $. The extension can be any number of characters but the system recognizes only the first two. An example of a file name with an extension is:

```
FOO.SV
```

The CLI often appends an extension to a filename to indicate the type of information the file contains and to distinguish it from other types of files created from the same source file. For example, assume that your source file is named A.SR. The CLI will append extensions to different versions of A, as follows:

A.RB relocatable binary file (after assembling source file).
A.SV core image (save file) (after loading or binding binary file)
A.LS listing file (if you specified listing file during the assembly step)
A.OL overlay file (if you specified overlays in your load or bind command)

As you develop your source programs into executable save files via the system assemblers and binders, you can ignore extensions if you give your assembly-language source files the extension .SR, or no extension. The utilities use a search algorithm to find the file with the appropriate extension. RDOS will always be given the extension .SV. The CLI gives the extension .SV to each executable program, but you need not enter this extension to execute the program; simply type filename) from the console.

If you append a unique extension to a filename, you must always append this extension to the filename when you want to access the file via the CLI or a system call. (Save files won't execute with an extension other than .SV.) For more on extensions, see the *CLI User's Manual.*

When you add your own extension to a file name, either avoid a CLI extension or use it properly. Don't, for example, confuse the operating system by giving a source file the extension .SV.

File Attributes and Characteristics

A file's attributes protect it; they permit or restrict reading, writing, renaming, deleting, or linking.

The attributes listed below apply primarily to disk files. To protect nondisk files, RDOS assigns certain attributes which you cannot change. (Of course, you can always write-protect a file on magnetic tape by removing the write-enable ring.) Use either the RDOS call .CHATR (Chapter 3) or the CLI command CHATR to alter the access attributes of a file.

P permanent file; no one can delete or rename a file while it has the P attribute.

S save file (core image). RLDR or BIND automatically assigns this attribute. No file can be executed without it.

W write-protected file, which no one can modify.

R read-protected file, which no one can read.

A attribute-protected file. The attributes of such a file cannot be changed. After the A attribute has been set it cannot be removed.

N no resolution permitted. This attribute prevents a file from being linked to.

? first user-definable attribute.

& second user-definable attribute.

Note that you can assign your own attributes to a file with the characters ? and &; you place them in bits 9 and 10 of the attributes word. They are described further under the .CHATR command, Chapter 3. You should avoid giving a file more restrictive attributes than it needs. Note, for example, that you cannot in any way delete a file with attributes AP (except by erasing the entire disk by a procedure called full initialization).

Disk file characteristics are determined when you create a file, and you cannot change them thereafter. The list of file characteristics is:

D This file is randomly organized (all save files have the D characteristic).

C This file is contiguously organized.

L This file is a link entry (which contains nothing, but points to another file).

T This file is a partition (all partitions also have the C characteristic).

Y This file is a directory (includes partitions and subdirectories).

The CLI command LIST allows you to obtain information from a file directory about one or more files.

File Transfer

You can copy a file from any device to any other device with the CLI command XFER. The XFER command transfers the contents of one file to another file. There are two arguments:

```
XFER sourcefile destinationfile
```

If you type:

```
XFER $PTR A)
```

a file named A is created on disk, and the contents of the paper tape mounted in the paper tape reader are transferred to it. (The symbol) represents a carriage return.) If you type:

```
XFER MYFILE YOURFILE)
```

disk file YOURFILE is created, and the contents of the file named MYFILE are copied to it.

Note that RDOS is a disk-based system, and most of its commands and calls work best on disk files (many *require* disk files). We recommend that you copy any nondisk file to disk before trying to edit, compile, assemble, load, bind, execute or debug it. If the file was previously DUMPed, LOAD it to disk; if it is not in DUMP format, XFER it to disk. I/O on a disk is always faster and easier than I/O on any other medium.

Disk Files

Your RDOS system can support two controllers for every type of disk drive Data General provides. Each controller (except for 6001-6008) can control up to four disk drives. The 6045 or 4234 controller can support both diskette and disk drives.

The primary unit in an RDOS disk file is the disk block, which contains 256 16-bit words (512 bytes). When you create a disk file, the system call or CLI command you use directs the system to organize the file in one of three ways: sequentially, randomly, or contiguously.

In a sequential file, the system reads disk blocks in logical sequence, one by one; it reserves either the last word or last two words (depending on the disk) for a pointer to the next block. RDOS always reads and writes sequential files in blocks via system buffers, which takes time. You create a sequential file with the system call .CREAT, or the CLI command CREATE.

In a random file, the system uses a file index to access any block; generally, it never needs more than two disk accesses to access a block. (Very large files may require more accesses.) RDOS uses all 256 words for data storage. You can read and write random file blocks via Direct Block I/O, without system buffering; this saves time. To create a random file, use the system call .CRAND or CLI command CRAND.

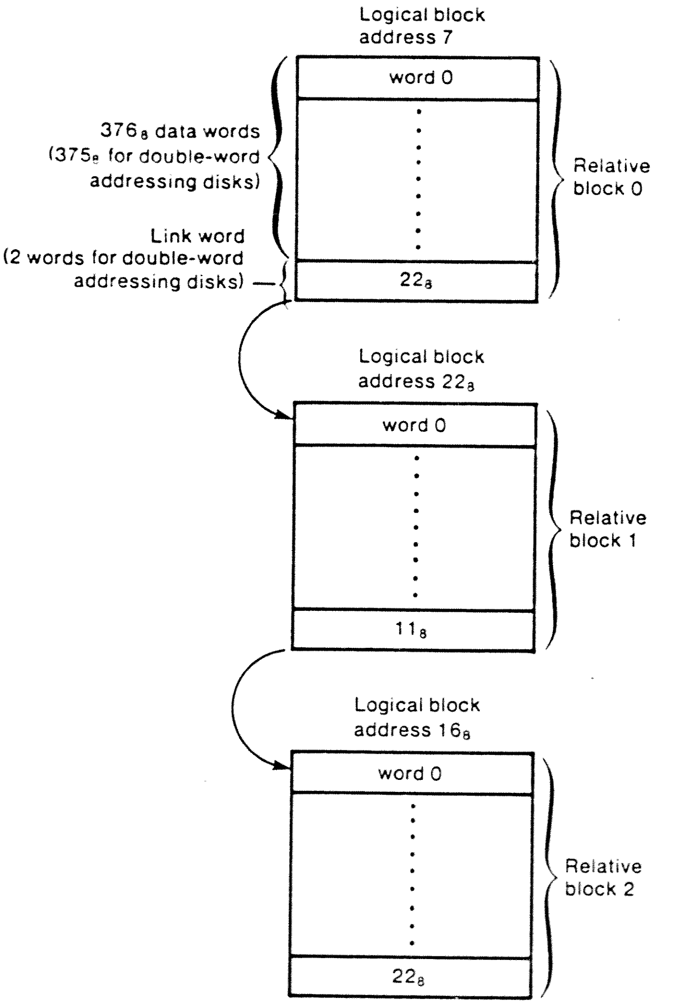
Contiguous file access is the fastest in RDOS. All blocks in a contiguous file are contiguous on disk; but unlike sequential and random files, each contiguous file has a fixed, unalterable length in blocks. This means that RDOS does not need a file index, and it needs only one disk access. Each block uses all 256 words for data storage. You can also use Direct Block I/O for a contiguous file. You create a contiguous file with system call .CCONT or .CONN, or CLI command CCONT. See Chapter 3 for the difference between .CONN and .CCONT.

RDOS offers you five ways to access disk files for I/O. In all but the last mode (called Direct Block I/O), RDOS transfers files via system buffers. See Chapter 3, *I/O Commands*, for I/O modes.

Sequentially Organized Files

When the system writes a sequential file to disk, the first block has relative number 0, the second 1, and so on. RDOS gives each block a *logical* address, and uses this address to derive the block's physical sector/track location on disk. To find the next relative block, it stores a link to the next block in the last word of the block (last two words on multiple-platter disks). This link is invisible to you, but not to RDOS, which uses it to compute the physical address of the next or previous relative block.

As an example, assume that RDOS is reading block 0 of a sequential file. When it reaches the link at the end, which contains block 1's logical address it then moves to block 1 and continues reading. Blocks 0 and 1 need not be contiguous on disk. From block 1, RDOS reads forward, but it can never skip a block; to reach block 7, it would have to read until it encountered the link at the end of block 6. Figure 2-1 shows this concept.


Logical block address 7
word 0
376_8 data words (375_8 for double-word addressing disks)
Link word (2 words for double-word addressing disks)
22_8
Relative block 0
Logical block address 22_8
word 0
11_8
Relative block 1
Logical block address 16_8
word 0
22_8
Relative block 2

Any link word is the block address of the previous block, XORed with the block address of the next block. Links for the first and last relative blocks are XORed with zero. (as there is no previous or next block, respectively).

SD-00534

Figure 2-1. Sequentially-Organized Disk File

Whenever you access a sequential file for I/O, RDOS transfers it via system buffers. Block by block, RDOS reads the file into a system buffer for the transfer.

When RDOS writes data into its system buffer area, it overwrites the oldest available buffer block first. When all buffers have been used, the least-recently used is the first to be overwritten. After RDOS has read a block into its buffers, you can read or write the block's records directly; no further disk access is required.

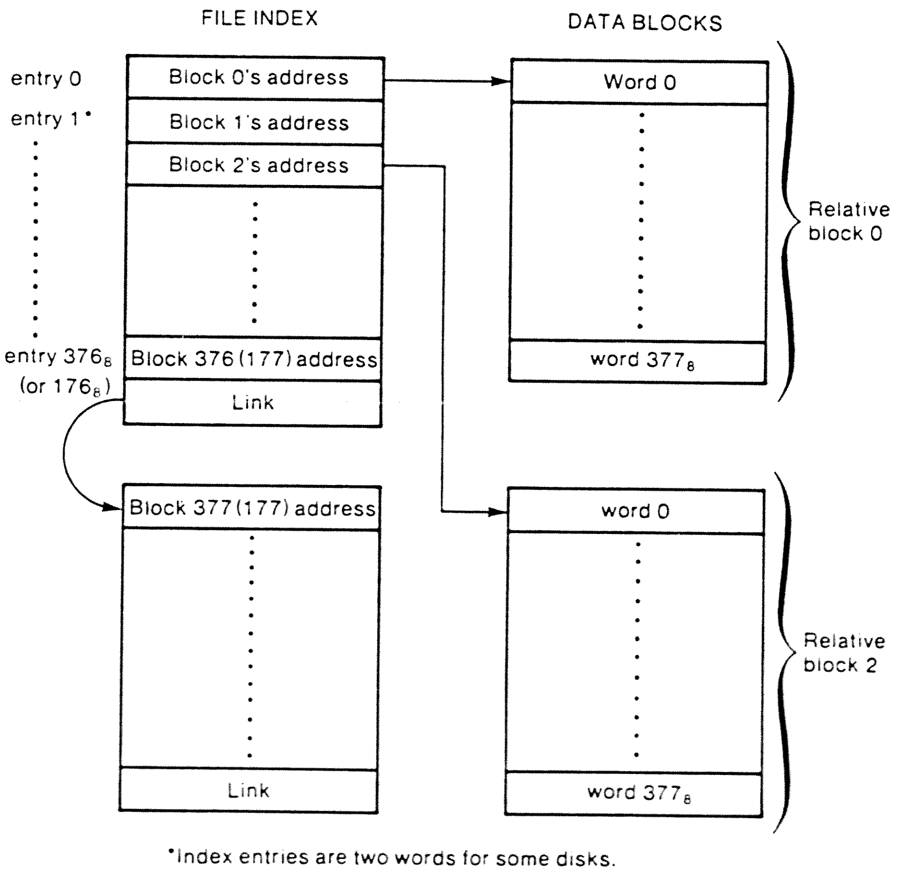
Randomly Organized Files

In RDOS, all save files employ random organization. When you create a random file, RDOS creates a file index for it. For each block you write in the file, RDOS enters one or two words (depending on disk size) in the file index; the index word(s) contains the block's logical disk address, which allows you to access any block on the disk. While index blocks are linked in the same way as sequential blocks, the last word or two words points to the next index block. The first data block in the file is number 0, the second 1, and so on; the first entry in the index is entry 0, and contains the logical address of block 0, and so on. If an index entry contains zeros (no address), then its corresponding block has not been written.

Figure 2-2 shows the relationship between the file index and data blocks in a randomly organized disk file.

For files which contain less than 255 data blocks, RDOS generally needs only two disk accesses to read or write a block: one for the file index and one for the block of data itself. If the file index is memory-resident (as it would be if you access the file previously and the index remained in a system buffer), only one access need be made.

If the data block itself is in memory, RDOS needs no disk accesses at all.



SD-00535

Figure 2-2. Randomly Organized Disk File

You can use all I/O commands available for sequential files on random files. Because random organization is more efficient, I/O is generally faster on random files. For large-scale I/O, you can shorten processing time even further by using Direct Block I/O commands to transfer your random files. In Direct Block I/O transfers, RDOS transfers an entire block from disk to the memory area you specify, without using system buffers. By avoiding buffering, you can save time, but you must manage records yourself; you lose the automatic management of the system buffers.

Contiguously-Organized Files

As shown in Figure 2-3, RDOS accesses data blocks in contiguously organized files randomly, without a file index. Contiguous files consist of a fixed number of disk blocks which are located at an unbroken series of disk block addresses. You can be neither expand nor reduce the size of these files. Since the data blocks are at sequential logical block addresses, all that RDOS needs to access a block within a contiguous file is the address of the first block (or the name of the file) and the relative block number within the file. RDOS organizes all disk partitions and overlay files contiguously.

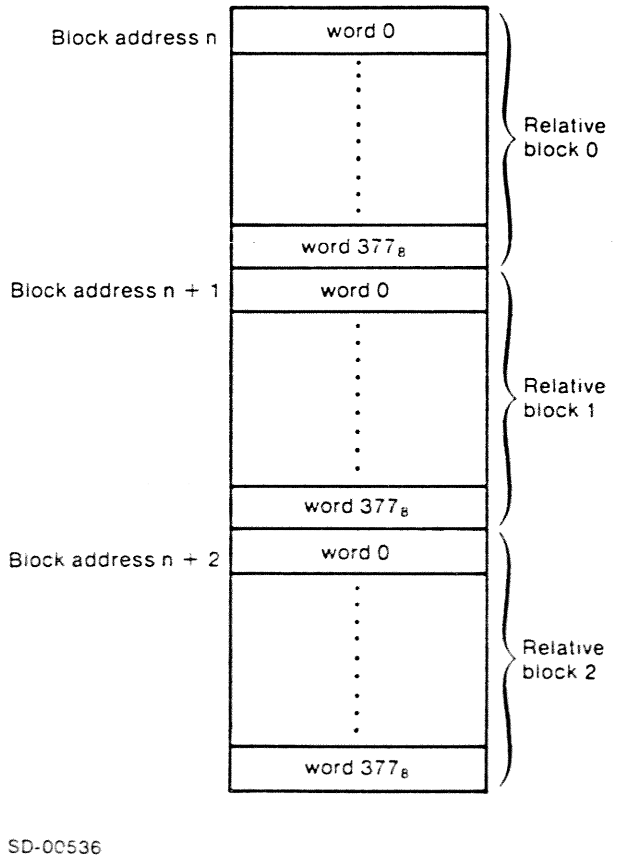


Figure 2-3. Contiguously Organized Disk Files

All I/O operations permitted on randomly organized files can be performed on contiguous files, but the size of the contiguous file remains fixed. Block access is faster in a contiguous file, since RDOS does not need to read a file index.

RDOS Disk Directories

Before you introduce a disk to the system, you must check and fully initialize it with the Disk Initializer program, DKINIT.SV. DKINIT.SV is a stand-alone program which you received with your system; it is further described in *How to Load and Generate Your RDOS System.* After DKINIT has run on the disk, you can elect to install a disk bootstrap on the disk; this will enable you to bootstrap an RDOS system on any other disk from this disk, as long as the new disk also contains the BOOT.SV program. The bootstrap occupies blocks 0 and 1 of the disk. The disk ID is in block 3, and the bad block pool created by DKINIT occupies block 4.

The first time you bring the disk into your system, RDOS creates on it two system directories, called SYS.DR and MAP.DR. SYS.DR records all file names and other file data on the disk; RDOS updates it whenever you create, modify, or delete a file or user directory on the disk. MAP.DR is a block allocation map; it records those blocks which are in use and those which are free for data storage. MAP.DR is aware of all disk space except blocks 0 through 5. Thus, these entries can never be destroyed since the system is unaware of the disk space where they reside.

Intitial Disk Block Assignments

As shown below, certain blocks on every disk have fixed assignments; the remaining blocks are free for system use or your file storage. Block 0 and 1 are reserved for the disk bootstrap program, BOOT; block 4 records bad disk blocks. Block 6 is the first index block of SYS.DR, the system directory. Block 7 is reserved for an index of file index blocks used whenever a program swap occurs. Blocks 10_8 through 16_8 are reserved for swap file indexes. Block 17_8 is reserved for the first block of the MAP.DR file.

Disk Block Number (octal)	
0,1	root portion of BOOT
3	disk ID
4,5	bad block pool
6	first index block of SYS.DR
7	index of file index blocks used for swap storage
8-16	swap storage index blocks
17-n	MAP.DR blocks (depend on disk size)
n+1-m	BOOTSYS.OL (always written by INIT/F)
: :	free blocks for RDOS or user files

The MAP.DR file starts at block 17_8; it is a contiguous file. Each bit of each word in MAP.DR indicates whether or not a specific block is in use, as follows:

Word	Contents
0 . . .	block allocation map, 1 bit for each block, from left to right in ascending order, starting with block number 6. 0 means that block is available, 1 means that block is in use.
n-1	'n' is the size of the partition in blocks/16 (integer division).

System Directory (SYS.DR)

You can create many directories within your RDOS system, and you can create files in each directory. RDOS writes a copy of SYS.DR to each directory, to keep track of the files within it. Each SYS.DR is a random file.

The system directory employs a hashing algorithm to speed up access of directory entries. RDOS allocates an initial system directory area at the time you initialize the disk with DKINIT.SV. This area (called a frame) is a contiguous set of disk blocks; to minimize head travel time. You can check and modify the frame size on a disk with DKINIT.

The first word in each block of SYS.DR is the number of files listed in the block. Following this word is a series of 22_8 -word entries, called user file descriptions or UFDs, which describe each file. Each block in SYS.DR looks like this:

Word (octal)	Contents
0	Number of files in this block of the directory (16_8 maximum)
1 . . . 22	User file description (UFD)
23 . . . 44	User file description (UFD)
. .	Remainder of block.
376_8	Contains maximum number of UFDs which ever existed in this block; if 16, indicates possible existence of overflow block.

The UFD describes the file's name, its two-character name extension, its size, its attributes and characteristics, the address of the first block, other qualities, and a logical code for the device which holds this file, as follows:

Word (octal)	Contents
0-4	Filename (padded with nulls, if necessary)
5	Extension (padded with nulls, if necessary)
6	Attributes and characteristics.
7	Link access attributes.
10	Number of last block in file.
11	Byte count in last block.
12	First address (physical address of first block in sequential or contiguous file; or first block of index for a random file).
13	Year and day last accessed.
14	Year and day created or most recently modified.
15	Hour and minute created or most recently modified.
16	UFD variable information.
17	UFD variable information.
20	Use count.
21	Device code DCT link.

The attributes in words 6 and 7 permit or restrict access to the file. See .CHATR and .CHLAT in Chapter 3 for more on this.

A nonzero file use count indicates that one or more users have opened the file. If a malfunction occurs when a file is open, its count will often be wrong; you must clear it to zero (via the CLI command CLEAR) before you can close, rename, or delete the file.

User Directories

Within any RDOS system, each user needs disk space for his files. Disk *partitions* and *subdirectories* permit you to organize and assign file space flexibly, by user or category name.

Although you can use either CLI commands or system calls to organize your disk space, we recommend using the CLI whenever possible. Error interpretation is faster and simpler through the CLI. After you have created the hierarchy you want from the console, you can access its directories and manipulate files via system calls in your programs.

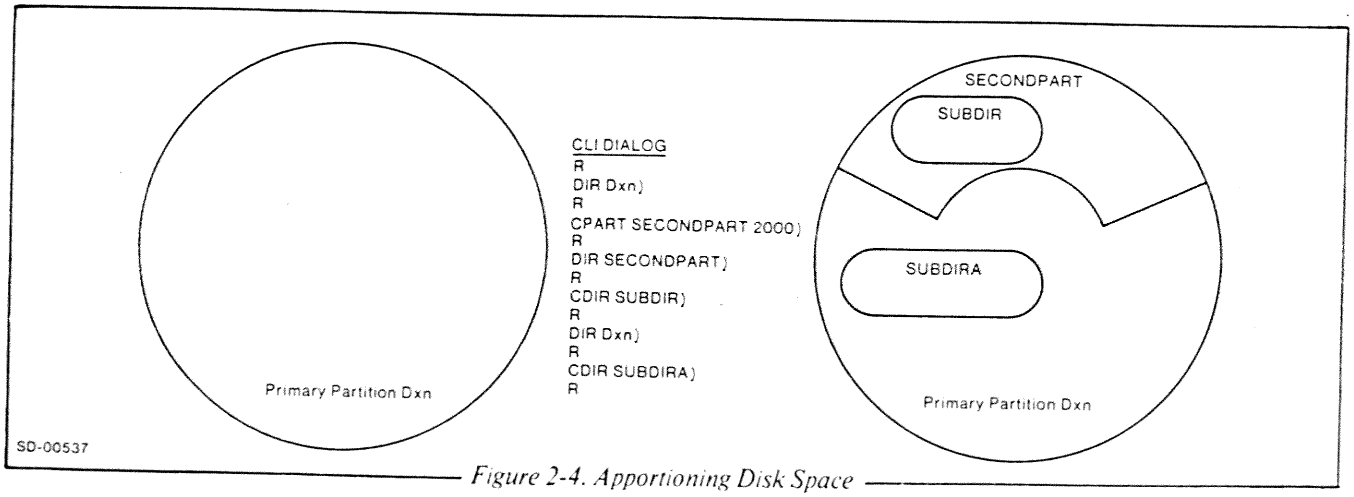
Partitions and Subdirectories

Each disk you introduce to the system contains a given number of blocks available for storage. These blocks make up an area called the *primary partition.* According to everyone's needs, you can logically detach sections of the primary partition and give them different filenames. These discrete sections are called *secondary partitions;* you create them and give them a fixed size with the CPART command (system call .CPAR). Within the primary partition (and secondary partitions, if any) are smaller groups called *subdirectories.* You create a subdirectory with the CDIR (.CDIR) command. Each subdirectory is flexible; it grows or shrinks according to the files you append or delete from it. A file can also exist in the master directory. A subdirectory and its files can never outgrow the fixed size of its parent partition.

A newly-created subdirectory consists of three blocks: SYS.DR's initial index block and data blocks for the SYS.DR and MAP.DR entries. The map directory entry in each subdirectory's SYS.DR is a copy of the MAP.DR entry in the parent partition.

In a multiuser RDOS system, the type of disk space anyone receives depends on the installation. Typically, each user has a personal directory, and unlimited reading access to several common public files. In some systems, each user has a large secondary partition for subdirectories and files; in others, each has a subdirectory on the primary partition.

Figure 2-4 below, shows a disk before and after partitioning; it also gives the CLI commands required to do the partitioning. DXn is a general term, which will vary according to your own disk(s) described in Table 2-1.



```
CLI DIALOG
R
DIR Dxn)
R
CPART SECONDPART 2000)
R
DIR SECONDPART)
R
CDIR SUBDIR)
R
DIR Dxn)
R
CDIR SUBDIRA)
R
```

Figure 2-4. Apportioning Disk Space

Each primary partition, secondary partition, and subdirectory contains a version of the disk's SYS.DR to keep track of the files within it, and enable it to access I/O devices. Each partition's SYS.DR also has a version of MAP.DR to maintain a record of free and occupied data blocks. Each subdirectory's SYS.DR uses a copy of its parent partition's MAP.DR.

One important advantage of secondary partitions is that a disk failure in a secondary partition won't affect files in other partitions. Other partitions' MAP.DRs aren't vulnerable to a failure. For this reason, some people prefer to place their systems and utilities in a secondary partition, and operate from that partition, using directory specifiers.

Partitions are contiguous files, and subdirectories are random files. They are unusual in that they contain other files, and receive the extension .DR -- but they are no more privileged than data files. You can dump, list, or load them; you can also delete all but the primary partition.

Initializing and Releasing a Partition or Subdirectory

You must initialize subdirectories and partitions before you can access the files or subdirectories within them. Intialization opens a subdirectory or partition, introduces it to the system, and prepares it for use. This procedure is called *partial initialization*. (*Full initialization* introduces new disks to the operating system; it writes a new SYS.DR, MAP.DR and BOOTSYS.OL on the disk, which effectively destroys all existing file structures).

When you have bootstrapped RDOS and completed the date/time log-on sequence, the CLI displays its R prompt. At this point, RDOS has initialized only the master directory, which holds the current RDOS system; this is often DP0, DP0F, DZ0, or DS0, but it can be another disk or secondary partition.

You can use either of two commands to initialize a subdirectory or partition: the CLI commands INIT or DIR (or system commands .INIT or .DIR).

```
INIT partition-or-subdirectory
```

While many partitions and subdirectories can be initialized at any moment, RDOS allows only one current directory at a time. The current directory is the one which RDOS searches for all files- unless you have told it to search elsewhere. The DIR command selects a new current directory and initializes it at the same time (if it hasn't already been initialized). For example:

```
DIR partition-or-subdirectory)
```

During system generation, you specify the maximum number of subdirectories and partitions which can be initialized at any moment. The current maximum is 64. If your INIT or DIR (or .INIT or .DIR) would exceed your system's maximum, you'll receive an error message (or your program will take the error return).

After you have initialized a directory, it is part of the system; RDOS will remember where it is, and access it even if it is on another partition or subdirectory. It remains in the system until you release it. To release a directory, type the CLI command:

```
RELEASE subdirectory-or-partition)
```

(or use the system call .RLSE). When you RELEASE a directory, you remove its initialization. If you release the current directory, the master directory becomes the current directory until you specify another current directory via DIR or .DIR. The master directory holds the operating system, and the system will shut down when you release it.

At shutdown, of course, you release the master directory via the CLI. You must release this directory before physically removing the disk which holds it (if this applies). If you are running two programs, you must do this from the background console, and you must terminate the foreground program before you do it. RDOS will then verify the release; e.g.,

```
RELEASE DP0)
```

MASTER DEVICE RELEASED

You can then turn off the computer, disk drive(s), and peripherals.

If you have more than one disk unit in your system, you will need to use a global directory specifier to initialize each one. Global specifiers are listed in Table 2-2; examples are DP0, DP0F (removable and nonremovable disks in unit 0, first top-loader controller), and DZ0 (first 6060-series unit).

For example, assume that you have just bootstrapped your system and that you have three disks: DP0, DP0F, and DZ0. The disk on which you bootstrapped RDOS would automatically become the current and master directory.

For runtime convenience, RDOS offers an equivalence command, EQUIV (or .EQIV). EQUIV (.EQIV) allows you to change the global specifier of any tape drive or disk (except the master device) before you initialize the device. This enables you to write programs without naming a specific disk or tape device. At run time, you select whatever device is available, and change its global specifier into your generic name via EQUIV, e.g.,

```
EQUIV DISK DP4)
```

You then initialize DP4 under its new name, and run your program. When you release the device, RDOS will restore its old specifier.

Referencing Disk Files

Because a file may exist in one of many subdirectories, and a subdirectory may reside in one of many partitions, your CLI command or system call must tell RDOS where to find the file. If you have more than one disk unit on your system, you may need to enter a *global* specifier (e.g., DP4) when you initialize the directory which holds the file.

Once you have initialized a directory (via INIT or DIR), you need not do it again; you need only enter the directory name, a colon, and the filename. For example, see Figure 2-4. Assume that you want to execute file MYPROG.SV, in subdirectory SUBDIR, on secondary partition SECONDPART. If you had initialized SUBDIR, you could simply type SUBDIR:MYPROG). But you haven't initialized SUBDIR. You can initialize any directory by entering the hierarchy names in descending order, separating each from the next with a colon, without spaces, e.g.,

```
INIT SECONDPART:SUBDIR)
```

You can also do this another way - by designating the directory you want as the current directory, via DIR (.DIR call). The command

```
DIR SECONDPART:SUBDIR)
```

initializes SUBDIR and makes it the current directory. All filename references without directory specifiers are directed to the current directory. Whenever you enter filenames including a colon specifier, RDOS assumes that you want a file in another directory, and makes a directory access there. For example, if your current directory had been SECONDPART (after the command DIR SECONDPART), you would have typed

```
INIT SUBDIR)
```

Naturally, if SUBDIR had been the current directory (after the command DIR SECONDPART:SUBDIR) you wouldn't need to INIT any directory - you could have executed MYPROG.SV by typing

```
MYPROG)
```

Master Directory

The master directory (device) on each disk has the following uses:

1. It becomes the current directory after you bring up the system, bootstrap a new system, or release a different current directory.
2. It contains the current RDOS system save and overlay files, and usually contains the system utilities and library, unless they were loaded into another directory or were never loaded or copied.
3. It contains push space for program swaps.
4. It contains the spool files, and tuning file (if any).

You determine the master directory when you bootstrap RDOS into operation; it remains the master until you release it, or until you bootstrap another system or program via the BOOT command.

Link Entries

The link entry allows a user in any directory to access any disk file or device, like MT0:n, $STTO1, or $LPT, by its name or by any other file name.

Link entries save disk file space by allowing users in different directories to access a single copy of a commonly-used disk file; this is their most popular application. Link entries may point to other link entries, with a depth of resolution of up to ten. The file which is finally linked to is called the resolution file. You can create a link entry with the CLI LINK command or the system command .LINK.

Creating a link entry is easy - the resolution file need not even exist when you do it. Your only requirement is that the link entry name be unique within its directory. The link entry can have the same name as the resolution file, or not; it can be on the same partition as the resolution file, or not.

The LINK command has two arguments:

```
LINK link-entry-name resolution-file-name
```

RDOS will create the link entry in the current directory, unless you specify another directory. RDOS assumes that the resolution file is in the current directory's parent partition (which can be either a secondary or the primary partition), *not* in a subdirectory. If the resolution file is elsewhere, you must indicate its location with colon specifiers.

The link entry need not have the same name as the resolution file. Link operations are clearer and simpler, however, if the link shares a name with its resolution file. Link entries with different names are called aliases.

To use a link, you (or your program) must initialize the directory containing the resolution entry and all directories containing intermediate link entries. Moreover, the attributes of the resolution entry, and all intervening link entries must allow linking (see .CHATR and .CHLAT, Chapter 3).

In Figure 2-5, two links exist to the resolution entry EDIT.SV on primary partition DP0. The resolution file - EDIT.SV - is the Text Editor supplied with your system. Normally, Data General utility programs are loaded onto the master directory before system generation, and EDIT.SV is included in these utilities. It is not in a subdirectory, and linking to it is easy.

The CLI command sequence which created the structure shown in DP0 is:

```
DIR SECONDPART)
R
LINK EDIT.SV EDIT.SV (or EDIT.SV/2)
```

The first EDIT.SV is the link entry name, the second is the resolution file name.

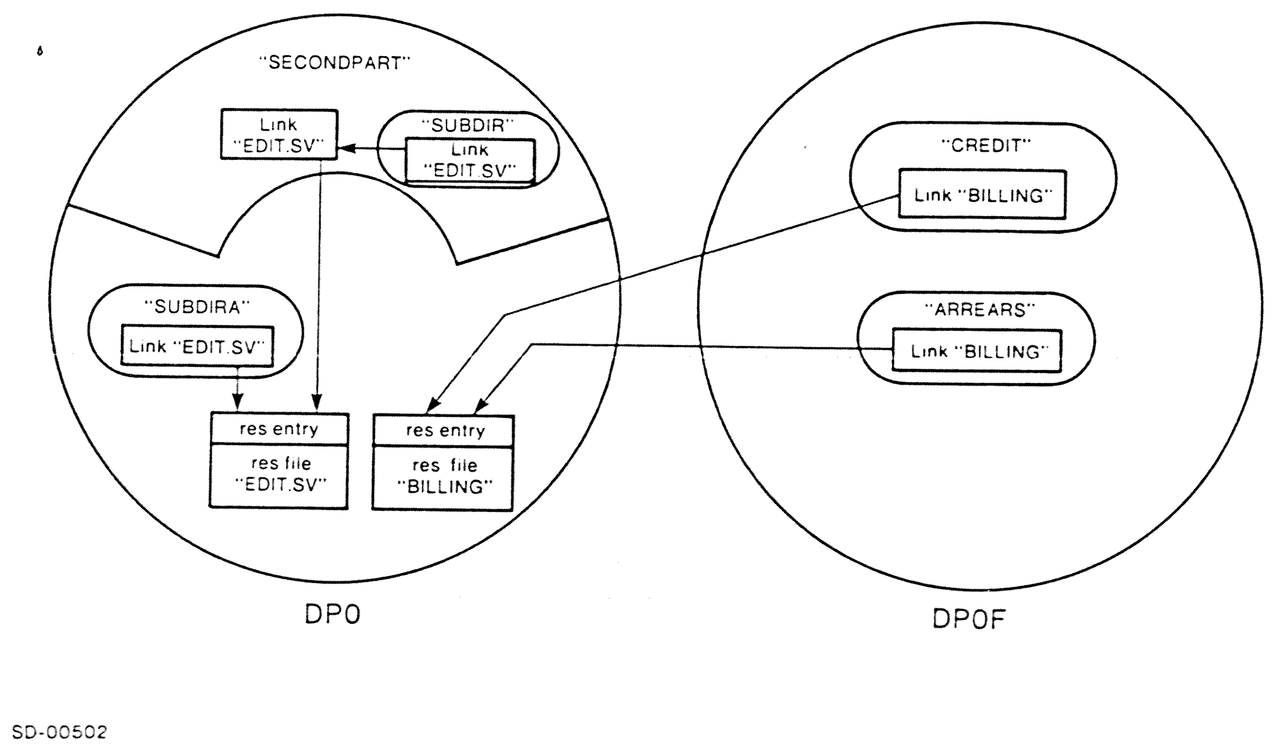


Figure 2-5. Link Entries

The LINK command created a link entry named EDIT.SV (in partition SECONDPART) to the editor on SECONDPART's parent partition, DP0. Now that SECONDPART is linked to EDIT.SV, any user in SECONDPART can use it to edit text while EDIT.SV occupies disk space on DP0 only.

The command sequence to link from subdirectory SUBDIR would be:

```
DIR SUBDIR)
R
LINK EDIT.SV DPO:EDIT.SV)
```

or

```
LINK EDIT.SV/2)
R
```

(For this chained link example, assume the second link command).

To link from SUBDIRA, you'd type:

```
DIR DPO:SUBDIRA)
R
LINK EDIT.SV/2))
R
```

To create the link entries from DP0F to file BILLING on DP0, you'd type:

```
DIR DPOF:CREDIT)
R
LINK BILLING DPO:BILLING)
R
```

and

```
DIR ARREARS)
R
LINK BILLING DPO:BILLING)
R
```

Once again, if the resolution file is not on the partition which holds the current directory you must input specifier information.

Before you can use a link, all immediate links must be resolvable. Thus you must initialize all intervening directories (if DP0 wasn't initialized in the example, neither link in DP0F would work). If you removed the link entry from SECONDPART (UNLINK or .ULNK), the link in SUBDIR would be useless but the link in SUBDIRA would still work. Note that UNLINK (.ULNK) is the only way to remove a link entry; if you try to DELETE (.DELET) a link, the link will persist and the resolution file will be deleted.

Each link entry is a filename, whose sole function is to point to the resolution entry (or to another link entry which is closer to the resolution entry). Like other files, each resolution entry has a user file definition which includes two sets of attributes: file access attributes (called resolution entry attributes) and link access attributes.

You assign resolution entry attributes to govern direct access to the file; you change the attributes via the CLI command CHATR or system command .CHATR (Chapter 3). The attribute N forbids linking (it actually allows the link, but prevents anyone from using it); other attributes govern reading, writing, renaming, or deletion. The A attribute makes all other attributes of a resolution entry or file permanent.

Link access attributes permit or restrict access to the resolution entry. Again, the N attribute forbids linking. You can use the CLI command CHLAT or system command .CHLAT to change these attributes.

Thus, although you can create a link to a resolution file very easily, two sets of resolution entry attributes guard the resolution file. As seen by a link entry, the resolution file has a composite of link attributes and resolution entry attributes.

More than one link entry may point to a resolution entry. Single user read-write opens and multiple read-only opens are allowed.

In any command or system call, using a link has the same effect as using the resolution file name. For example, in Figure 2-5, assume that the current directory is CREDIT on DP0F. The following sequence of CLI commands is the same as CRAND DPO:RATINGS

```
LINK RATINGS DPO:RATINGS)
CRAND RATINGS
```

After either set of commands, the current directory remains CREDIT, and file RATINGS exists on DP0.

After you create and link a file, you cannot open the resolution file (by a system *.OPEN* command) until you have initialized all directories in the path to the resolution file. The system will return error ERDNI (Directory Not Initialized) or error ERDSN (Device Not In System) from the OPEN command if you haven't initialized all intervening directories.

Link Devices

The link entry offers much more than a simple way to share user files. You can create a link entry for any file - including a reserved device such as the line printer.

If you establish a link to a mag tape or a cassette resolution file, you must initialize the device before the link will work. You cannot link a nondisk device in turn to another resolution file.

File Access Example

When you introduce a new disk to the system, only its primary partition exists. At this point, you can choose a directory structure for the disk, according to the partition and subdirectory definitions at the beginning of this chapter.

For example, assume that six users need space on one disk for their files. Ideally, each user would have as much disk space as needed, yet file space would be used efficiently, and each user's files would be safe from unauthorized access or alteration. There are two obvious plans for dividing the disk:

Plan 1

Create six secondary partitions, and assign one user to each partition.

Plan 2

Create a single secondary partition (six times as large as each secondary partition in Plan 1). Assign each person to a distinct subdirectory within the partition.

In both Plan 1 and Plan 2, everyone's disk files would be protected and each person would be able to access files in the primary partition (like utility programs). Plan 1 guarantees a fixed amount of file space to each person. If one person exhausts his or her space, he/she cannot appropriate unused space on another person's partition. Plan 2 allows each person to grab as much file space as he/she requires from within the common secondary partition, as long as there is any unused file space. Under plan 2, no one has a guaranteed minimum amount of file space at any moment, although file space is used more efficiently than in plan 1.

The best solution for this sample installation involves a middle ground: one secondary partition for two hungry users, another secondary partition for a prolific, hungry user, and subdirectories and files for 3 modest users. Commonly-used public files will remain on the primary partition; users can link to them from their directories. Figure 2-6 shows this flexible solution; a sample dialog with this system's CLI follows the figure.

In the illustration the symbol ⌓ means secondary partition, ⬭ means subdirectory and ▭ means data file, and ↓ means link entry.

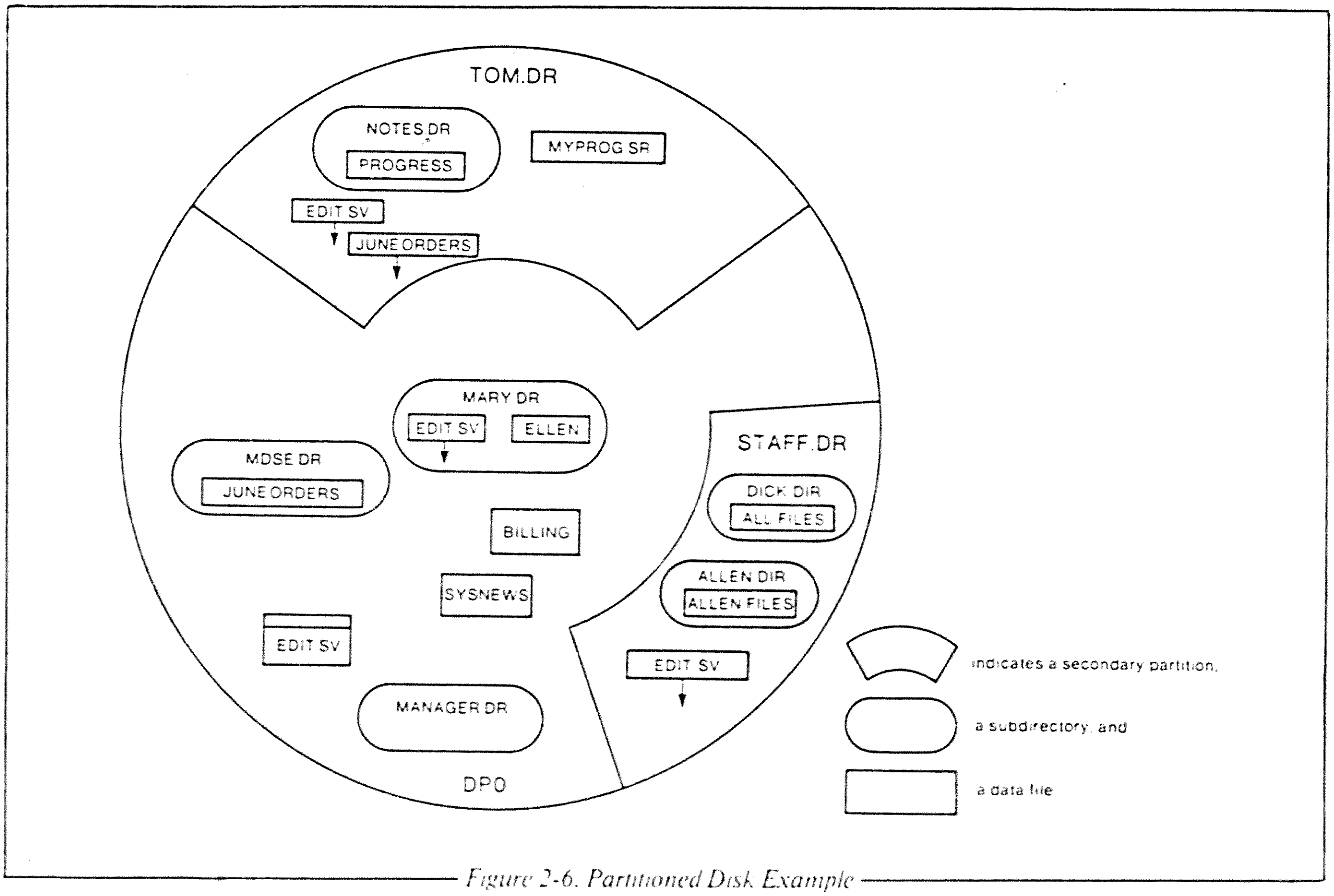


Figure 2-6. Partitioned Disk Example

```
bootstrap sequence ..........

FILENAME? )

log-on sequence ..........

R

PRINT SYSNEWS)
```

After the bootstrap and log-on sequences, the CLI announces itself; the master directory automatically becomes the current directory. The PRINT command prints the contents of file SYSNEWS on the line printer.

```
R
PRINT MARY:ELLEN)
NO SUCH DIRECTORY:MARY:ELLEN
R
INIT MARY)
R
PRINT MARY:ELLEN)
```

To RDOS, a directory which hasn't been initialized doesn't exist. The INIT command opens directory MARY; PRINT prints file ELLEN. DP0 remains the current directory.

```
R
GDIR)
DP0
R
DIR TOM:NOTES)
NO MORE DCBS:NOTES
R
RELEASE MARY)
R
DIR TOM:NOTES)
R
```

In this example, the GDIR command returns the name of the current directory (DP0); the operator then tried to initialize partition TOM and subdirectory NOTES. Unfortunately, this RDOS system was generated to allow only 3 partitions and subdirectories to be initialized at any moment. DP0 and MARY were initialized, and TOM and NOTES would have brought the total to 4; hence the error message. Releasing MARY made room for TOM and NOTES; NOTES became the current directory.

```
R
PRINT PROGRESS)
```

prints file PROGRESS found in subdirectory NOTES.

```
R
PRINT TOM:MYPROG.SR)
```

We need the directory specifier, since MYPROG.SR is not in the current directory, NOTES.

```
R
DIR TOM; LINK EDIT.SV/2)
R
LINK JUNEORDERS MDSE:JUNEORDERS)
R
RELEASE TOM
R
```

(You can enter several CLI commands on one line if you separate them by a semicolon.) The first command creates a link entry in TOM to the Text Editor utility program on DP0. The second LINK command creates a link entry named JUNEORDERS to file JUNEORDERS, in subdirectory MDSE. Now we can reference both the editor and JUNEORDERS through partition TOM, although they occupy significant amounts of file space on partition DP0 only.

```
R
INIT MT0)
R
DUMP/V MT0:0 STAFF)
FILE DOES NOT EXIST: STAFF
R
DIR DP0:STAFF)
R
DUMP/V MT0:0
        ALLFILES
        *DICK.DR
        ALLENFILES
        *ALLEN.DR
        EDIT.SV


R
DIR DP0)
R
LOAD/V MT0:0)
        ALLFILES
        *DICK.DR
        ALLENFILES
        *ALLEN.DR
        FILE ALREADY EXISTS: EDIT.SV
R
```

This sequence initializes drive MT0 and dumps the contents of STAFF to file 0 of the tape mounted on tape unit 0. The /V switch requests verification of the files dumped. The next sequence makes DP0 the current directory, and loads the dumped files from MT0:0 into DP0. Again, the /V switch requests verification. The link entry EDIT.SV is dumped but can't be loaded, because filename EDIT.SV exists on DP0.

```
R
DIR DP0; RELEASE STAFF)
R
DELETE/V STAFF.DR)
DELETED STAFF.DR

R
RELEASE DP0)
MASTER DEVICE RELEASED
```

After loading subdirectories HARRY and ALLEN onto DP0, the operator deletes their parent partition, STAFF.DR. He could have deleted them individually, but he saved a step by deleting STAFF. The space STAFF occupied returns to partition DP0. The session ends with the release of the master directory, DP0.

Directory Command Summary

The following list summarizes the CLI and .SYSTM commands used to manage disk files and directories; see Chapter 3 and the *CLI Reference Manual* for more information about these commands.

CLI Command	System Command	Meaning
CCONT	.CCONT	Create a contiguous file with all words zeroed.
CDIR	.CDIR	Create a subdirectory.
CHATR	.CHATR	Change file attributes.
CHLAT	.CHLA	Change link access entry attributes.
CLEAR		Set a file's use count to zero.
	.CONN	Create a contiguous file without zeroing words.
CPART	.CPART	Create a secondary partition.
CRAND	.CRAND	Create a random file.
CREATE	.CREAT	Create a sequential file.
DELETE	.DELET	Delete a file.
DIR	.DIR	Specify a new current directory, initialize it if necessary.
EQUIV	.EQIV	Assign a new name to a global directory specifier, removing the old name or system name.
INIT	.INIT	Initialize and open a directory or device.
LINK	.LINK	Create link entry to a file in any any directory.
RELEASE	.RLSE	Remove a directory or a device from the system.
RENAME	.RENAM	Rename a file.
UNLINK	.ULNK	Delete a link entry.

Magnetic And Cassette Tape Files

You can access data on magnetic tape and cassette by both file I/O and free form I/O. RDOS permits file access on nine- and seven- track magnetic tape, and supports up to 16 magnetic tape and 16 cassette tape drives. For free form I/O, the tape controller supports reading and writing at any density; file I/O requires high density if on a dual-density drive.

The following are the I/O modes generated by the operating system:

Tape File I/O:	7-track 800BPI, EVEN Parity 9-track NRZI800BPI, ODD Parity 9-track PE 1600BPI, ODD Parity
Free Form I/O:	Parity in any hardware combination except WRITE EOF is always EVEN for 7-track, ODD for 9-track.

If a controller detects an error during reading, the system will attempt to reread the data 10 times before issuing error code ERFIL, "file data error." If a data error is detected and returned to the CLI, the system will dislay the message: PARITY ERROR: FILE MTn:dd, for mag tape, or CTn:dd for cassette, where *n* is the unit number and *dd* represents the file number.

If RDOS detects an error after writing, it will attempt to backspace, erase, and rewrite up to ten times. If the rewrite fails the tenth time, then you will get the error message.

If RDOS receives an undefined error, it will return the tape status word as the error code. If it returns this code to the CLI, you will see it as: UNKNOWN ERROR CODE n, where *n* is the tape status word.

Nine and Seven Track Data Words

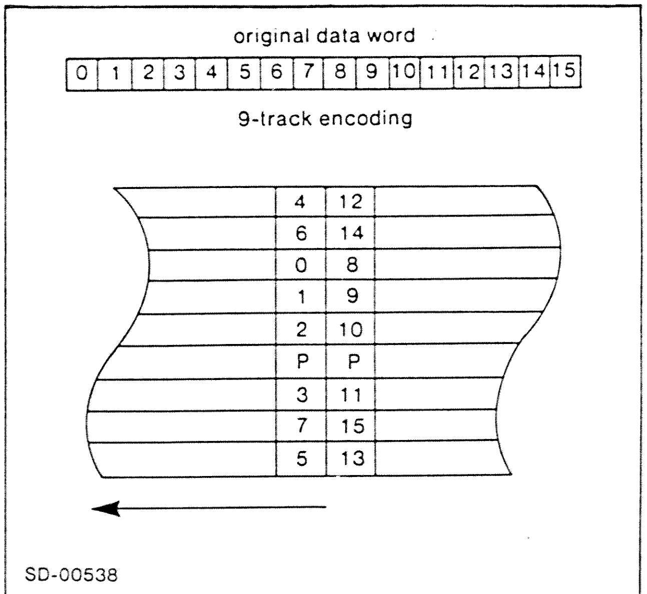
Each data word output to nine track units, under both file I/O and free format I/O, is written as two successive eight-bit bytes. Data is encoded as in Figure 2-7.

Data output to seven-track units is necessarily encoded in tape file I/O. RDOS encodes each 16-bit word as 2 data words, in 4 successive frames. In free form I/O, RDOS encodes each word as 2 successive frames. (See Figure 2-8.)

Each tape has a physical end-of-tape (EOT) marker. Whenever you attempt to write beyond this marker, RDOS will return the error ERSPC after it completes the operation . You cannot start a new file beyond the physical end of tape marker.

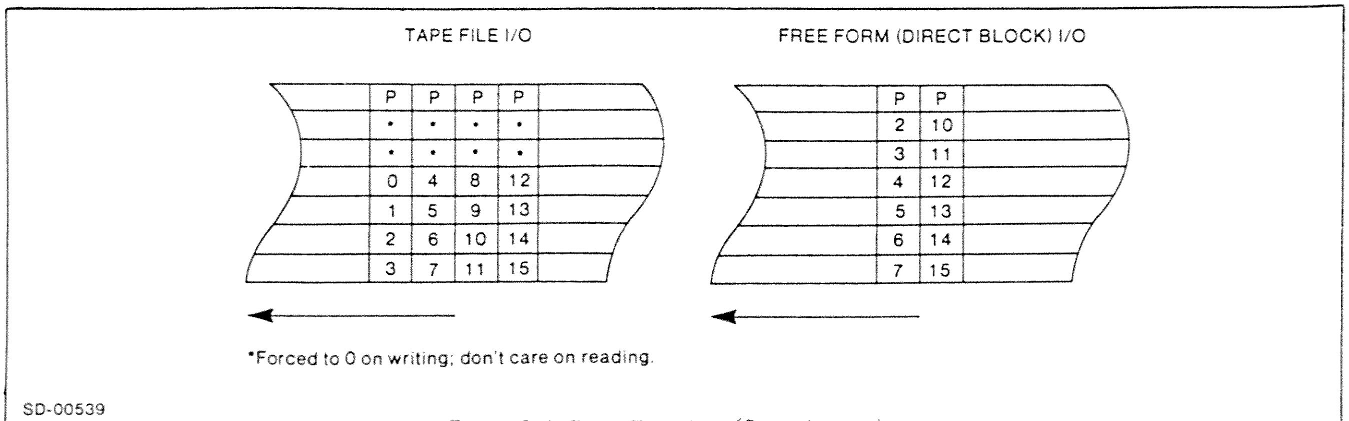
If you are writing to tape via the CLI DUMP command, the system will stop writing and abort the command when it reaches the EOT mark. If you are writing on a system level, make sure that the reel holds enough tape to accept the file. (The error mnemonic for EOT is ERSPC; you can use this to terminate writing before running the tape from its reel.)

Upon reaching the physical EOT while writing, you should terminate the tape file to avoid running the tape from its reel.



SD-00538

Figure 2-7. Data Encoding (9-track units)



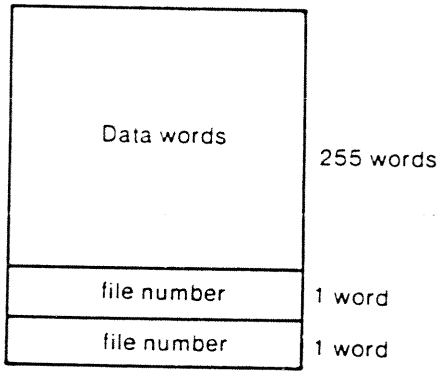
•Forced to 0 on writing; don't care on reading.

SD-00539

Figure 2-8. Data Encoding (7-track units)

Tape File Organization

In tape file format, RDOS writes and reads data in fixed-length blocks of 257 16-bit words. It fills short blocks with nulls. Data files are variable in length, and each one contains as many fixed-length blocks as you need. The first 255 words of each block contain your data, and the last two words each contain the file number. The following illustration shows the structure of a data block:



SD-01032

After the first file, RDOS writes a double end-of-file (EOF) mark. The system begins writing at the first double EOF it finds, overwrites the second EOF in the pair, writes the file, and signifies the end by writing another double EOF. RDOS writes files in consecutive order, starting with file number 0 and extending through file number 99.

Initializing and Releasing a Tape Drive

To initialize a tape drive, use the CLI INIT command; e.g., INIT MT0). INIT automatically rewinds the tape on that drive. Full initialization (INIT/F) rewinds the tape and writes two EOFs (the logical end-of-tape indication) on the beginning of the tape. You must always perform an INIT/F on all new mag tapes before you use them. Note that INIT/F effectively erases the tape by permitting the system to overwrite all files on it.

The CLI RELEASE command rewinds a tape and releases its drive from the system.

Referencing Tape Files Using File I/O

Files are placed on tape in numeric order, beginning with file number 0. If a tape is long enough, you may place up to 100 files on it; the last file will have number 99.

To access a tape file in a command line, enter the command and the tape specifier, followed by a colon and a file number. For example:

```
PRINT MT0:6)
```

MT is the specifier for magnetic tape, 0 is the drive unit number, and 6 is the file number. The format and definitions of all magnetic tape specifiers are:

MTn:m or CTn:m
Mag tape or cassette unit n, where n is a number between 0 and 17_8 and has no leading zero; file number m is in the range 0-99.

You need not enter a leading zero to enter the first 10 file numbers. To reference file number 8 of the tape on magnetic tape unit 2, you would use either of the following:

```
MT2:08 or MT2:8
```

You must enter both the tape global specifier and the file number. Violation of this rule will cause the system to respond: ILLEGAL FILE NAME.

Some examples of references to files on tape and disk are:

```
DUMP MT0:0)
```

Dump all nonpermanent file onto tape from the current directory (this provides a magnetic tape backup). The files become file number 0 of the tape mounted on unit 0.

```
LOAD MT0:0)
```

Reload the files in tape file 0 into the current disk directory.

```
XFER $PTR CT2:9
```

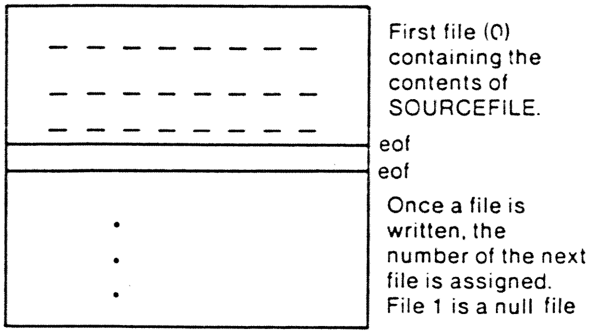
Transfer the contents of the file in the paper tape reader to file 9 of the cassette mounted in cassette drive 2.

Note that only files which have been DUMPed via CLI can be LOADed onto disk; you must XFER any file which is not in DUMP format.

You must write files on magnetic tape in numeric order. For example, assume that you transfer a disk file to tape unit 0. Tape unit 0 contains a new tape, which you have just fully initialized.

```
XFER SOURCEFILE MT0:0)
```

SOURCEFILE becomes the first file on the new tape, which contains the following:



The system recognizes only file numbers 0 and 1 on the tape; because RDOS assigns numbers incrementally, only these numbers exist.

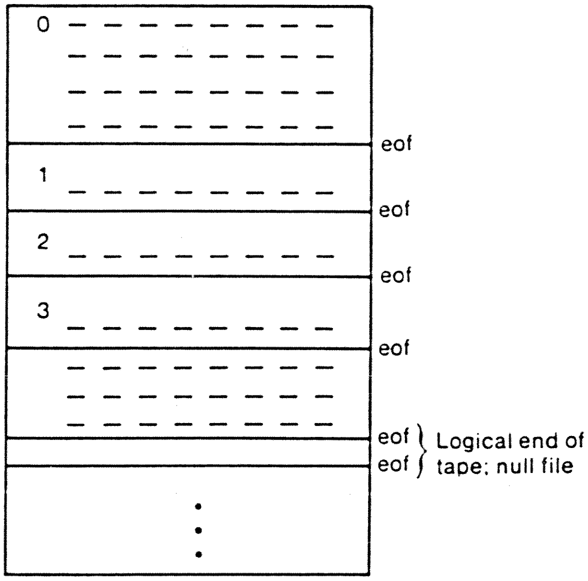
If you try to reference any other file on the tape:

```
XFER MYFILE MT0:2)
```

The system will be unable to find file 2, because file 0 is the last file. You will get the error message:

```
FILE DOES NOT EXIST: MT0:2
```

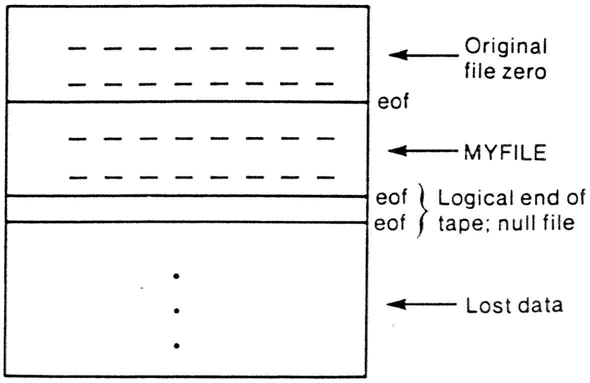
As you write files on tape, you should note their numbers. Otherwise, you could inadvertantly overwrite a file, and thus destroy the overwritten file and all following files. For example, assume a tape on drive 0 contains four files:



The command:

```
XFER MYFILE MT0:1)
```

overwrites the contents of file 1 with MYFILE, and voids the location data of following files. The original file 1 and all subsequent files are lost:



Before you physically remove a mag tape, reel or cassette, you must RELEASE its transport. This command rewinds the tape and resets the system tape file pointer to file 0, for correct file access in the future. (If you forget to do this, simply RELEASE the drive).

You must also note the implications of the logical end-of-tape mark (double EOFs) employed by RDOS. For example, if you deliberately write a null file, you cannot write any other files to the tape. Your null file will be the last file.

Linking to Tape Files

You can link tape files from disk files with the mechanism described under disk files. Linking disk file A to tape file MT0:0 creates a link entry in the current directory for resolution file MT0:0; the link entry to file MT0:0 is named A. References to file A in the current directory are resolved as references to file MT0:0.

Free Form Tape Reading and Writing

In addition to tape file I/O, which uses 257-word blocks, RDOS allows you to read and write data to magnetic tape in free format, record by record. You open a tape unit for free form I/O with .MTOPD, and write or read the data with .MTDIO; these calls are described in detail in Chapter 3, under *Input/Output Commands.*

Essentially, .MTDIO allows your program to read or write from 2 to 4096 words within a data record, and to space forward or backward through one to 4095 data records or to the start of a new data file. Additionally, this call allows your program to rewind a reel, write an end-of-file mark, read the transport status, and perform other machine-level operations. Unlike tape file I/O, the system does not maintain a tape file pointer under free form I/O after it locates the file you specified in .MTOPD.

Multiplexors

The SYSGEN program allows you to specify multiplexors and their characteristics. RDOS supports several kinds of Data General multiplexors: the type 4255-4258 Asynchronous Line multiplexor (ALM-device codes 34_8 for the primary ALM, 44_8 for the secondary ALM) and the type 4060-4063 Asynchronous Communications multiplexor (called QTY: device code 30_8 for QTY, 70_8 for QTY1). Either the ALM or QTY can support from one to 64 full- or half-duplex lines. RDOS also supports a Universal Line Multiplexor (ULM). A ULM can support 16 half-duplex asynchronous lines or 8 full-duplex asynchronous lines and/or two synchronous lines. RDOS does not support the synchronous lines (other software available with RDOS, like the Communications Access Manager, does support them). A full-duplex line allows data to flow two ways simultaneously: users can transmit to RDOS over it, and RDOS can transmit to users' terminals. RDOS assumes full-duplex lines, but you can set up half-duplex protocols if you want.

Each ALM, ULM, or QTY line is a filename, of the form QTY:x where *x* is a number from 0 to 63. You can open multiplexed lines on any RDOS I/O channel (channels are described in Chapter 3). After you have opened a line on a channel, you can use system calls .RDL/.WRL and .RDS/.WRS to read and write to it. In Chapter 3, *I/O Channel Numbers* describes selecting a channel number, and *Input/Output Commands* describes opening a file, and the read/write calls. No more than one read and one write can be outstanding on any one line. To close a line, and abort I/O, you must .CLOSE its channel (because the .ABORT task call doesn't affect QTY/ALM I/O).

When you .OPEN a multiplexed line (or any file), the contents of AC1 determine what operations RDOS will allow on that line. AC1 acts as a characteristic disable mask, as described under .OPEN (Chapter 3). The following characteristic bits affect multiplexors:

AC1	Meaning
DCCRE=1B4	Masking disables carriage return echoes on line reads (CR then acts as enter key).
DCLAC=1B6	Masking disables line feed after CR.
DCPCK=1B7	Masking disables software parity on QTY; no effect on ALM or ULM.
DCXON=1B8	Masking enables XON/XOFF protocol for STTR. (This prevents the teletypewriter reader from overflowing the multiplexor read buffer.)
DCNAF=1B9	Masking disables 20 nulls after line feed.
DCKEY=1B10	Masking disables echo, CTRL Z end-of-file, and line and character rubout.
DCTO=1B11	Masking enables backspacing for rubout (CRT displays only).
DCLOC=1B13	Masking makes this a modem line.
DCCGN=1B14	Masking disables TAB expansion.
DCNI=1B15	Masking enables multiplexor interrupts.

When AC1 equals 0 on the .OPEN, the multiplexed console has the following default characteristics:

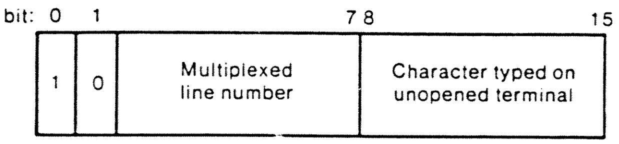
1) line feeds after carriage returns
2) 20 nulls after line feed
3) during line reads: characters are echoed. SHIFT-L (\) deletes line. RUBOUT deletes character and is echoed as —. CTRL Z results in end-of-file error. ESCAPE also results in end-of-file error.
4) this is a local line.
5) TABS are expanded as spaces.

Checking Multiplexed Lines for Activity or Interrupts

Line 64 Reads

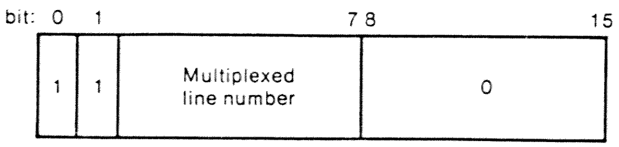
RDOS allows you to monitor both activity on all unopened multiplexed lines and console interrupts from all opened multiplexed lines. If a task opens QTY:64, and issues a read line or read sequential call, RDOS will suspend this task until someone either presses a key at the end of an unopened line, or hits an interrupt on an opened line. When RDOS receives the character typed, it readies the task, takes the normal return from the read call, and passes the following data in AC2:

A



When RDOS receives and answers a ring from a *modem*, it will send the following data to line 64, in AC2:

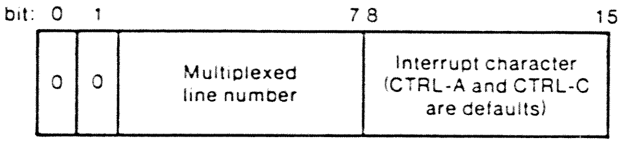
B



This allows your program to detect a service request from a distant terminal. If the request comes from an unopened line, your program can then .OPEN the line for communications. QTY:64 can be opened by both a foreground and background task; if this happens each task will receive characters from unopened lines.

If an *open* line receives an interrupt (CTRL A and CTRL C are defaults), RDOS will ready the task which .OPENed line 64, and pass the following data in AC2:

C



At SYSGEN, you can select interrupts other than CTRL-A and CTRL-C.

A task receives and interrupts from an *opened* line only if it is in the ground that opened the line's channel.

Line 64 Writes (ALM and ULM only)

RDOS allows you to change the device characteristic disable mask, line speed, or modem state on any ALM line. Just issue a .WRL to a channel opened on QTY:64, and pass the following data:

To change the mask (on .OPENED lines only):
AC0 = W64DC + line number
AC1 = new mask

To change line speed:
AC0 = W64LS + line number
AC1 = new line speed (0,1,2, or 3 for ALM clock; 1 through 15 for ULM line code, as described below)

To change modem state:
AC0 = W64MS + line number
AC1 = *[W64DTR][+][W64RTS]*

(Entries in italic brackets are optional.) W64DTR raises Data Terminal Ready; if you omit it, DTR is lowered. W64RTS raises Request To Send; if you omit it, RTS is lowered.

To change any or all characteristics on any line:
AC0 = W64CH + line number
AC1 = new characteristic mask

These symbols are defined in the user parameter file PARU.SR. Appendix B contains a PARU listing.

Note that RDOS does not check the validity of your input, so be careful when you change the characteristics of an open line.

ULM Line Codes

At SYSGEN, you select a line speed for all ULM lines. You can change the line speed of any ULM line via the QTY:64 mechanism described earlier. Simply specify one of the codes below (decimal) in AC1 to select the matching line speed.

This code:	Selects this line speed:
1	19200
2	50
3	75
4	134.5
5	200
6	600
7	2400
8	9600
9	4800
10	1800
11	1200
12	2400
13	300
14	150
15	110

Multiple Channels

A ground can have several channels opened to the same line, but the same line cannot be opened in both grounds (except line 64).

The first channel opened on a line becomes the master channel, and all other channels opened on it become subordinate; if you close the master, the subordinate channel numbers will be unable to use the line. Before you can reassign (.OPEN) the subroutine channel numbers on another line, you must close each one. If you .OPEN a new channel on a line after .CLOSEing the master, the new channel becomes the master channel.

ALM and ULM Modem Support

The ALM and ULM support modems with the following six signals:

DTR - Data Terminal Ready (set either by RDOS or yourself)

RTS - Request to Send (set either by RDOS or yourself).

DSR - DataSet Ready

RD - Ring Detect

CD - Carrier Detect (this signal is handled by the hardware).

CTS - Clear To Send (handled by the hardware)

When you bootstrap RDOS, it raises DTR and RTS, unless you have changed the ALM parameter file (ALMSPD.SR) to specify low DTR and/or RTS. On a ring interrupt, RDOS raises both DTR and RTS. On a disconnect, if DSR is low, it lowers both DTR and CTS.

When a modem's DSR (DataSet Ready) is low, it cannot communicate; RDOS will take the error return on all reads/writes to its modem line, and it will place code ERRDY in AC2. Note however, that the error return occurs only if you defined the line as a modem line by masking DCLOC on the .OPEN.

Multiplexor Error Messages

The following errors relate to reads/writes on multiplexed lines. For other read/write errors, see .RDL/.RDS or .WRL/.WRS in Chapter 3. On the error return, AC2 may contain one of the following codes:

AC2	Mnemonic	Meaning
24	ERPAR	Parity error detected on read.
47	ERSIM	Duplicate read or duplicate write.
127	ERRDY	Line not ready: modem's DSR is low.
130	ERINT	Console interrupt received.

The following errors clear the read buffer and error the read request:

131	EROVR	Hardware overrun error on read.
132	ERFRM	Hardware framing error on read.

ALMSPD.SR

The source file, ALMSPD.SR, defines the line characteristics of each line of the ALM or ULM. You can edit this source file and assemble it with MAC (a macroassembler) to tailor your multiplexed lines for specific applications. You can then generate a new RDOS system, during which SYSGEN will include the new ALMSPD.RB. If you do not define a line in this module, or if you set its characteristics at default, then it has the following characteristics:

1) clock frequency (ALM) or line speed (ULM) as set in SYSGEN
2) 1 stop bit
3) 7 bits per character
4) even parity
5) no loopback
6) DTR+RTS raised on initialization

You can define these characteristics for any line by inserting the line

```
LNDEF xx,DEFAULT
```

in ALMSPD.SR where xx is the two digit decimal number for the line you want to set. If you want to define different line characteristics, then insert a line of the form

```
LNDEF xx,spd,stop,bits,par,loop
```

or

```
LNDEF xx,spd,stop,bits,par,loop,dtr,rts
```

where

xx is the two-digit decimal line number;

spd is the clock frequency (may be 0,1,2, or 3 for ALM clock or 1 through 15 for ULM line speed);

stop is the number of stop bits per character (may be 1 or 2);

bits is the number of bits per character (may be 5, 6, 7, or 8, not including the parity bit);

par defines whether you wish no parity to be generated or checked (specify NO), even parity (EVEN), or odd parity (ODD);

loop tells whether you want to enable loopback (specify LOOPBACK or NOLOOPBACK);

dtr defines the state of Data Terminal Ready on initialization (DTRHIGH or DTRLOW); and

rts defines the state of the Request To Send on initialization (RTSHIGH or RTSLOW).

Note that you may omit the arguments for dts and rts if you wish to set their states as high.

For an ALM example, to set line 3 to have clock frequency 1, 2 stop bits, 7 bits per character, EVEN parity, and no loopback, you'd insert the following line. Both Data Terminal Ready and Request To Send will be initialized high.

```
LNDEF 03,1,2,7,EVEN,NOLOOPBACK
```

For a ULM example, to set line 4 to run at 4800 baud, 1 stop bit, 7 bits per character, ODD parity, and no loopback, you'd insert the following line. Both Data Terminal Ready and Request To Send will be initialized high.

```
LNDEF 04,9,1,7,ODD,NOLOOPBACK
```

After defining ALMSPD.SR, type MAC ALMSPD SLPT/L before performing a new RDOS SYSGEN.

End of Chapter

Chapter 3
Single Task Programming

This chapter describes most of the system calls you will need to program in RDOS in a single-task environment. It explains system and task command structures, summarizes the most commonly-used system calls, and then lists complete descriptions of single-task calls, under the headings:

DEVICE AND DIRECTORY COMMANDS
FILE MAINTENANCE COMMANDS
LINK COMMANDS
FILE I/O COMMANDS
CONSOLE I/O COMMANDS
MEMORY ALLOCATION COMMANDS
DEVICE ACCESS COMMANDS
CLOCK/CALENDAR COMMANDS
SPOOLING COMMANDS
KEYBOARD INTERRUPT COMMANDS

For important single-task material on program swaps and overlays, read the beginning of Chapter 4; for user interrupts read Chapter 7. If you want to run two grounds in your system, read Chapter 6, and, if you have a mapped system, the last half of Chapter 4. Chapter 5 covers tasks and multitasking; it includes system clock commands which you can use in a single-task environment.

Multiple and Single Task Environments

A program task is an execution path through user address space which uses system resources such as I/O, overlays, or simply CPU control. User address space includes all memory from location 16_8 through NMAX-1.

In a single-task environment, the program itself is the only task. A program creates a multitask environment by creating a task via task calls .TASK or .QTSK. If you plan a multitask program, you must specify multiple tasks either with assembly-language pseudo-ops or with RLDR switches. If you do so, RLDR will copy the multitask scheduler (called TCBMON) into your program, and allot the number of Task Control Blocks (TCBs) specified.

If you omit both task and I/O channel pseudo-ops, and task/channel switches, RLDR assumes a single-task program, and copies the single-task scheduler into your program. RLDR also allots eight channels for the program - enough for most single-task programs. Either a single or multitask program can use all system calls in this chapter. For more on multitasking, see Chapter 5.

Note that the task scheduler and other modules differ from each type of system (e.g., unmapped NOVA and mapped NOVA), which means that programs loaded under one type of system will probably not execute on another type of system. To load for a different system, obtain the proper system library (SYS.LB) for the target system, and ensure that RLDR searches it, not the current library, during the load. You can do this by loading from a subdirectory which contains the target system library and links to RLDR.

System and Task Calls

RDOS system and task calls allow you to communicate directly with the operating system. System calls and task calls are similar, but not identical.

You begin each *system* call with the mnemonic .SYSTM, which assernbles as a JSR @ 17 instruction. This instruction enables the system to respond to your command.

After the system has obeyed a system call, it takes a normal return to the second instruction after the command word. If it detects an exceptional condition, it takes the error return to the first instruction following the command word. System calls always reserve AC2 for the error code.

The general form of a system call description is:

ACn - Required input to the call

.SYSTM
command
error return (error code in AC2)
normal return (each accumulator, except AC3, is restored unless it is used to return output)

ACn - Output from the call

AC3 - The contents of location 16 (the User Stack Pointer) is the default value.

There are two basic types of system calls: those which require a channel number, and those which don't. Channel numbers are described below.

Many system calls require you to include a byte pointer to a specific filename. When you include this byte pointer, you can include a directory specifier as well. All RDOS system calls are summarized in Table 3-1, below.

A *task* call resembles a system call, with these exceptions:

1. You enter no .SYSTM mnemonic before the task command word.
2. RDOS executes task calls in user address space, not in system space.
3. *Task* calls which cannot take an error return do not reserve an error return location. Almost all system calls reserve an error return location even if no error return is possible. The commands in this chapter are *all*system calls.

See Chapter 5 for more detail on the differences between system and task calls.

Table 3-1. System Call List

.APPEND	Open a file for appending.
.BOOT	Bootstrap a new system. Chapter 8.
.BREAK	Interrupt the current program; save the current state of memory in save file format.
.CCONT	Create a contiguously organized file with all data words zeroed.
.CONN	Create a contiguously organized file with no zeroing of data words.
.CDIR	Create a subdirectory.
.CHATR	Change file attribues.
.CHLAT	Change link access attributes.
.CHSTS	Get the status of the file currently open on a specified channel.
.CLOSE	Close a file.
.CPART	Create a secondary partition.
.CRAND	Create a random file.
.CREAT	Create a sequential file.
.DDIS	Disable user access to a device in a mapped system.
.DEBL	Enable user access to a system (mapped) device.
.DELAY	Delay the execution of a task.
.DELET	Delete a file.
.DIR	Change the current directory.
.DUCLK	Define a user clock.
.EOPEN	Open a file for reading and writing by one user only.
.EQIV	Assign a temporary name to a device.
.ERDB	Read one or more disk blocks into extended memory (mapped). Chapter 6.
.ERTN	On an error, return from program and describe error (if to CLI).
.EWRB	Write one or more 256-word blocks from extended memory to disk (mapped). Chapter 6.
.EXBG	Checkpoint a background program (mapped). Chapter 6.
.EXEC	Swap or chain in a new program. Chapter 4.
.EXFG	Execute a program in the foreground. Chapter 6.
.FGND	See if there is a foreground program running. Chapter 6.
.GCHAR	Get character from the console.
.GCHN	Get the number of a free channel.
.GCIN	Get the operator input console name.
.GCOUT	Get the operator output console name.
.GDAY	Get today's date.
.GDIR	Get the current directory name.
.GHRZ	Examine the real time clock. Chapter 5.
.GMCA	Get the current MCA unit number. Chapter 8.
.GPOS	Get the current file pointer.
.GSYS	Get the name of the current operating system.
.GTATR	Get file attributes.
.GTOD	Get the time of day.

Table 3-1. System Call List (continued)

.ICMN	Define a program communications area, Chapter 6.
.IDEF	Identify a user device.
.INIT	Initialize a device or a directory.
.INTAD	Define a program interrupt task.
.IRMV	Remove a user device, Chapter 7.
.LINK	Create a link entry.
.MAPDF	Define a window map (mapped). Chapter 6.
.MDIR	Get the logical name of the master device.
.MEM	Determine available memory.
.MEMI	Change NMAX.
.MTDIO	Perform free format I/O on tape or cassette.
.MTOPD	Open a mag tape or cassette for free format I/O.
.ODIS	Disable keyboard interrupts for this console.
.OEBL	Enable keyboard interrupts for this console.
.OPEN	Open a file for reading and/or writing by one or more users.
.OVLOD	Load a user overlay into memory.
.OVOPN	Open a user overlay file.
.OVRP	Replace an overlay file.
.PCHAR	Write a character to the console.
.RDB	Read one or more disk blocks.
.RDCMN	Read a message from the other program's communications area. Chapter 6.
.RDL	Read a line.
.RDOPR	Read an operator message. Chapter 5.
.RDR	Read a random record.
.RDS	Read sequential bytes.
.RDSW	Read the console switches.
.RENAM	Rename a file.
.RESET	Close all files.
.RLSE	Release a directory or device.
.ROPEN	Open a file for reading only by one or more users.
.RSTAT	Get a resolution file's statistics.
.RTN	Return from a program to a higher-level program. Chapter 4.
.RUCLK	Remove a user clock. Chapter 5.
.SDAY	Set today's date.
.SPDA	Disable spooling.
.SPEA	Enable spooling.
.SPKL	Delete the current spool file.
.SPOS	Set the current file pointer.
.STAT	Get a file's statistics.
.STMAP	Set the data channel map for a user device (mapped), Chapter 6.
.STOD	Set the time of day.
.TUOFF	Turn the tuning report function off. Chapter 9.
.ULNK	Delete a link entry.
.UPDAT	Update the current file size.
.VMEM	Determine the number of memory blocks.
.WRB	Write one or more 256-word blocks to disk.
.WRCMN	Write a message to the other program's communications area. Chapter 6.
.WREBL	Remove the write protection of a memory area. Chapter 6
.WRL	Write a line.
.WROPR	Write an operator message.
.WRPR	Protect a memory area (mapped). Chapter 6.
.WRR	Write a random record.
WRS	Write sequential bytes.

Status On Return From System Calls

Status of the accumulators upon return from the system (.SYSTM or task call) is as follows: if the system returns no information as a result of the call, the carry and all accumulators except AC3 are preserved. For certain calls, the system returns information in AC0, AC1 and/or AC2.

By default, on return from any system call, AC3 contains the contents of location 16_8, (the USP), unless you specified a given module in the RLDR command line, as shown below. On an error return, RDOS uses AC2 to return a numeric error code. Appendix A lists the error codes.

Note: In this chapter, and throughout the manual, error codes listed under each call represent the most common errors only; the meanings have been expanded and interpreted in light of the call.

AC3 on Return

If you loaded your program with module:	then (upon return from call) AC3 contains contents of:
NSAC3 (any machine; used by default)	USP (location 16_8).
N3SAC3 (NOVA3s only)	Frame Pointer register.
ESAC3 (ECLIPSEs only)	Frame Pointer (location 41_8)

I/O Channel Numbers

Before you can access a file for I/O, you must give it an I/O channel number in your open call. While the file is open, it retains this channel number, and you must access it via the number instead of the filename. When you close the file, the number is released. The number immediately follows the call word in your program; if the channel number is n, the I/O calls for a file could run:

```
open n
      .
      .
      .
file reads/writes n
      .
      .
      .
close n
```

In a mapped system, you specify the maximum number of foreground and background channels during SYSGEN; the maximum for each ground is 377_8. In an unmapped system, SYSGEN asks no question about channels and the maximum for the system is 377_8.

For a single-task program, RLDR allots eight I/O channels, numbered 0 through 7. Usually, this is enough. If you want to specify more channels, use either the RLDR /C switch, or the assembler pseudo-op .COMM TASK.

Selecting a Channel

There are two ways to assign a channel number to a file: either directly when you open, e.g.,

```
.OPEN 3
```

or via AC2. If you specify number 77 (or CPU) on your open, RDOS will open the file on the channel number contained in the right byte of AC2. To open on a number *above* 77 (assuming that your program permits one), you must open on 77 and pass the number in AC2. The major advantage to opening on 77 is that you can use system call .GCHN to find a free channel for your open.

.GCHN returns the number of a free channel in AC2, and you can give this number a name, and use the name for all I/O to the file. This method ensures a free channel for file I/O (unless all channels are in use). Here is an example:

```
.SYSTM
.GCHN
JMP ER
STA 2, FILE1        ;STORE THIS CHANNEL
.                   ;NUMBER UNDER "FILE1".
.
.
LDA 2, FILE1
.SYSTM
.OPEN 77            ;OPEN "FILE1" FOR ANYTHING.
JMP ER
.SYSTM
.WRS 77             ;WRITE TO "FILE1".
.
.SYSTM
.GCHN
JMP ER
STA 2, FILE2        ;STORE NUMBER UNDER "FILE2".
.SYSTM
.APPEND 77          ;OPEN "FILE2" FOR APPENDING.
JMP ER
.
.
.
```

Capsule Command Summary

As you write different programs for your application, you will use certain system calls quite frequently, and others rarely or not at all. The following table attempts to summarize the most useful calls, in the sequence which you might use in a program. It gives the call name, format, accumulator data, and possible error codes. It assumes that you will use the CLI to create and initialize partitions and subdirectories, to execute mag or cassette tape I/O, and to control spooling, and that your program won't do such esoteric things as alter file attributes, create link entries, or manage a multitask environment. Of course, you can do all of these things via RDOS system calls if you choose.

The summary also assumes a single-task environment; it does not cover foreground/background calls (Chapter 6) or multitasking (Chapter 5). Many commands not given below are included in the rest of this chapter (or manual). Each call has the form:

```
.SYSTM
call name
error return to program
```

for example:

```
                LDA 0,BTPTR
                .SYSTM
                .DIR
                JSR ERROR
                .
                .
                .
BTPTR:          .+1*2
                .TXT "DP1:SUBDIR"
ERROR:          .SYSTM
                .ERTN
                JMP.-2
```

Each file I/O command requires a channel number, as noted. The term "btptr" means byte pointer.

Table 3-2. Common Call Summary

Call	Purpose	Remarks
.CRAND	Create a random file.	AC0:btptr to filename.
.CCONT	Create a contiguous file.	AC0:btptr to filename. AC1:number of disk blocks for the file.
.OPEN n	Open a file for I/O on channel n.	AC0:btptr to filename. AC1: characteristic disable mask. You can specify the system default mask (normal procedure) by passing 0 via a SUB 1,1 instruction before the .OPEN.
.APPEND n	Open a file for appending, on channel n. Set position for writing at the end of the file.	AC0:btptr to filename. AC1: characteristic disable mask. As with .OPEN you can use the default the .APPEND.
.RDL n	Read an ASCII line on channel n. Counterpart of .WRL.	AC0:btptr to area large enough for line (133 maximum). AC1 returns the count of characters read.
.RDS	Read sequentially from the file .OPENed on channel n. Sequential mode is required for binary data.	AC0:btptr to starting byte address of data. AC1:number of bytes to be read. to be read.
.WRL n	Write an ASCII line to the file OPENed on channel n. Writing begins at start of file if you .OPENed the file; at end if you .APPENDed the file. Limit is 132 characters, terminated by a CR, null, or form feed.	AC0:btptr to area which holds the ASCII line.
.WRS n	Write sequential bytes to the file on channel n. See .WRL for position information.	AC0:btptr to starting byte address of data. AC1:number of bytes to be written.
.WRB n, .RDB n	Direct-block I/O calls. Write or read a series of disk blocks to or from the random or contiguous file on channel n.	AC0:starting address for the block write or read. AC1:starting relative block number in the series. AC2:left byte-number of 256-word blocks to be written or read to the file.
.CLOSE n	Close the file. opened on channel n. RDOS then updates the file's UFD infor- mation. (.ERTN and .RTN close all chan- nels in the current program.)	
.DELET	Delete a file.	AC0:btptr to filename.

The following calls control NMAX, execute and return from program swaps or chains, and load overlays.

.MEM	Return the current program's NMAX value in AC0, and the value of the Highest Memory address avail- able for user programs in AC1.	
.MEMI	Raise NMAX to the value entered in AC0, or lower NMAX by the value entered in two's complement in AC0. RDOS returns the new value in AC1.	
.ERTN or .RTN	Close all channels in the current program and return to (resume execution of) the next higher-level program (usually the CLI). .ERTN returns an error code in AC2; if return is to the CLI, it also prints an error message on the console.	
.OVOPN n	Open overlay file for reading, on channel n. Before your program can use overlays, you must open them on a channel. You close the channel via a .CLOSE n.	AC0:btptr to overlay filename, including .OL extension.
.OVLOD n	Load an overlay from the overlay file opened on channel n into its reserved memory node.	AC0:overlay descriptor. AC1:conditional load flag.

If your program takes the error return from any of the calls above, AC2 will contain one of the following error codes:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number (legal range: 0 through 377_8).
1	ERFNM	Illegal filename (only alphanumeric or $ characters are permitted).
3	ERICD	Illegal command for device (for example, trying to read from the line printer).
6	EREOF	End of file detected while reading; or attempt to write beyond the end of a contiguous file.
7	ERRPR	The file is read-protected.
10	ERWPR	The file is write-protected.
11	ERCRE	The file already exists.
12	ERDLE	The file (directory) does not exist.
13	ERDE1	The file cannot be deleted because it has the permanent attribute.
15	ERFOP	The file hasn't been opened.
21	ERUFT	This channel is in use.
22	ERLLI	Line limit (132 characters) exceeded.
26	ERMEM	Attempt to allocate more memory than is available.
27	ERSPC	Current partition file space exhausted.
33	ERRD	Attempt to read or write into system space (unmapped systems only).

AC2	Mnemonic	Meaning
36	ERDNM	Device not in system.
37	EROVN	Illegal overlay number.
40	EROVA	File not accessible by direct-block I/O.
47	ERSIM	Simultaneous reads or writes attempted to same QTY/ALM line.
52	ERIDS	Illegal directory specifier.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space (mapped systems only).
101	ERDTO	Disk timeout occurred.
103	ERMCA	This MCA channel is in use.
104	ERSRR	A short receive request terminated the MCA transmission.
106	ERCLO	MCA/QTY/ALM output terminated by channel close.
124	ERZCB	Attempt to create a contiguous file of zero length.

Device and Directory Commands

This section describes the RDOS system commands which pertain to opening and releasing disks, mag tape and cassette drives, and disk directories; it also covers disk partition and subdirectory creating commands. It includes these commands:

.INIT	Initialize a directory/device.
.DIR	Select a different current directory.
.RLSE	Release a directory/device.
.GDIR	Get the current directory's name.
.CDIR	Create a subdirectory.
.CPART	Create a secondary partition.
.EQIV	Temporarily rename a nonmaster device or tape drive.
.GSYS	Get the current RDOS system's name.
.MDIR	Get the master directory's name.

Commands for individual files are covered in the following section, *File Maintenance.*

RDOS can support many directory devices simultaneously. During SYSGEN, you configured your system for specific disk and tape devices, and you can address any of these by its name as shown in Table 2-1.

Initialize a Directory or Device (.INIT)

Your program can initialize devices and directories via the system command .INIT.

If AC1 contains anything but -1 when you invoke .INIT. a partial initialization of the device or directory results; this makes all files in the directory available to the system software. Partial initialization of a magnetic tape or cassette rewinds the tape and resets the tape file pointer to file zero. If AC1 contains 177777 when you invoke .INIT, a full initialization of the device results. Full initialization on a mag tape or cassette rewinds the tape and writes two EOF's to signify the logical end-of-tape. You lose all files on that tape. Full initialization of a disk builds a virgin SYS.DR and MAP.DR effectively destroying all existing files. RDOS treats full initialization of a secondary partition or subdirectory as a partial initialization.

Required input

AC0 - Byte pointer to a directory/device specifier.

In each byte pointer, bits 0-14 contain the word address which holds or will receive the byte. Bit 15 specifies which half (0 left, 1 right).

Format

```
.SYSTM
.INIT
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
10	ERWPR	Device is write-protected. (full initialization only).
12	ERDLE	Directory does not exist.
27	ERSPC	Out of disk space.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
45	ERIBS	Insufficient number of Device Control Blocks (DCBs), specified at SYSGEN time.
51	ERNMD	Insufficient number of Device Control Blocks specified at SYSGEN.
52	ERIDS	Illegal directory specifier.
56	ERDIU	In a dual processor system, using an IPB, the other CPU is using this directory.
57	ERLDE	Link depth exceeded.
74	ERMPR	Address outside address space.
77	ERSDE	Error detected in SYS.DR of nonmaster device.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
112	EROVF	Too many chained directory specifiers caused system stack overflow. This can occur only when links are used in the specifier string.
121	ERFMT	Disk format error. Try to dump the disk, and run DKINIT on it.
122	ERBAD	Disk has invalid bad block table (for action, see 121 above).

Change the Current Directory (.DIR)

When you bootstrap an RDOS system, the directory which holds the system becomes the current directory. The .DIR command selects a different current directory -- if the new current directory hasn't been initialized, .DIR will also initialize it.

After you .DIR to a directory, you can access all files in it without using directory specifiers.

.DIR is not mandatory for file access in nonmaster directories, because RDOS permits directory specifiers in all filename arguments to system commands. For example, both of the following arguments would access MYFILE in DP4, from master directory DP0F:

```
1.        .TXTM 1
             .
             .
          LDA 0,   .MYFILE
             .
             .
.MYFILE:  .+1*2
           .TXT "DP4:MYFILE"

2.        .TXTM 1
             .
             .
          LDA  0, .DP4
          .SYSTM
          .DIR
             .
             .
          LDA  0, .MYFILE
             .
             .
.DP4:      .+1*2
           .TXT "DP4"
.MYFILE:   .+1*2
           .TXT "MYFILE"
```

In the first example, DP0F remains the current directory; in the second, DP4 becomes the current directory.

Required input

AC0 - Byte pointer to directory name string.

Format

```
.SYSTM
.DIR
error return
normal return
```

If RDOS takes the error return, the current directory definiton remains unchanged.

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	Directory does not exist.
27	ERSPC	Out of disk space.
36	ERDNM	Device or directory not in system.
51	ERNMD	Attempt to initialize too many directories at one time (not enough DCB's specified at SYSGEN).
52	ERIDS	Illegal directory specifier.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
74	ERMPR	Address outside address space
101	ERDTO	Disk timeout occurred.
112	EROVF	System stack overflow due to excessive number of chained directory specifiers.
121	ERFMT	Disk format error. Try to DUMP the disk, and run DKINIT.SV on it.
122	ERBAD	Disk has invalid bad block table. See 121 for action.

Release a Directory or Device (.RLSE)

This command dissociates a directory or device from the system, and prevents further I/O with it.

You should always release a removable disk via either the CLI command RELEASE (or .RLSE) before removing it from the unit. You must also close all files within a directory before you can release it. Release of a master directory releases all directories. The master directory is the directory which holds the current RDOS system. You can get its name by using the .MDIR call or the MDIR command.

Required input

AC0 - Byte pointer to a directory or device specifier.

Format

```
.SYSTM
.RLSE
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
56	ERDIU	Directory in use.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
114	ERNIR	Attempted release of a tape unit containing an open file.

Get Current Directory Name (.GDIR)

This call returns the name of the current directory or device (e.g., DP0). This name is followed by a null; it doesn't include the names of superior directories, or colon specifiers. For current directory *DP0F:PART2:DIR1*, it would return *DIR1*.

Required input

AC0 - Byte pointer to 13_8 -byte area to receive the current directory/device name.

Format

```
.SYSTM
.GDIR
error return
normal return
```

The first 12_8 bytes will contain the name (with trailing nulls, if necessary); byte 13_8 will contain a null terminator.

Possible errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read into system area.
74	ERMPR	Address outside address space.

Create a Subdirectory (.CDIR)

This call creates an entry for a subdirectory name in the current partition's system directory (SYS.DR). The subdirectory will automatically receive the .DR extension.

Required input

AC0 - Byte pointer to the directory name (directory specifiers permitted).

Format

```
.SYSTM
.CDIR
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal directory name.
11	ERCRE	Attempt to create an existent directory.
53	ERDSN	Directory specifier unknown.
55	ERDDE	Attempt to create a subdirectory within a subdirectory.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Create a Secondary Partition (.CPART)

This command creates an entry for a secondary partition name in the current SYS.DR. The secondary partition will automatically receive the .DR extension.

Required input

AC0 - Byte pointer to secondary partition name.

AC1 - Number of contiguous disk blocks in secondary partition (the minimum is 60_8). RDOS allocates disk blocks in integer multiples of 20_8; if your number is not an integer multiple of 20_8, the system will truncate it to the next lower multiple.

Format

```
.SYSTM
.CPART
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal secondary partition name.
11	ERCRE	Attempt to create an existing secondary partion.
46	ERICB	Insufficient number of free contiguous disk blocks available.
53	ERDSN	Directory specifier unknown.
54	ERD2S	Partition too small (must have at least 60_8 blocks).
55	ERDDE	Attempt to create a secondary partition in a secondary partition (i.e., a tertiary partition).
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Assign Temporary Name to Disk or Tape Unit (.EQIV)

This command assigns a temporary name to a disk or tape unit, permitting unit independence during the execution of your program. Thus you might write all mag tape references in your program as MTAPE, and, at run time, use the .EQIV command to assign the name MTAPE to a specific device (e.g., MT6). You must issue this command before you initialize the device (under its new name). You cannot assign a temporary name to the master device.

A device keeps a temporary name until you release it; it then reverts to its old specifier. You can then .EQIV (CLI command EQUIV) another name before initialization, if you want.

Required input

AC0 - Byte pointer to current global specifier name.

AC1 - Byte pointer to temporary name.

Format

```
.SYSTM
.EQIV
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
53	ERDSN	Directory specifier unknown.
56	ERDIU	Device in use (i.e., already initialized).
74	ERMPR	Address outside address space.

Get the Current Operating System Name (.GSYS)

This call returns the name of the currently-executing operating system, its .SV extension, and a null terminator.

Required input

AC0 - Byte pointer to 15_8 -byte area.

Format

```
.SYSTM
.GSYS
error return
normal return
```

The first 12_8 bytes will contain the name (with trailing nulls, if necessary); byte 13_8 will contain a null terminator.

Possible errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read or write into system area.
74	ERMPR	Address outside address space.

Get the Name of the Master Directory (.MDIR)

Because you can bootstrap an RDOS system in a secondary partition, the master directory might not have an obvious disk name, like DP0. .MDIR returns the name of the master directory.

Required input

AC0 - Byte pointer to 13_8 byte area to receive the directory name.

Format

```
.SYSTM
.MDIR
error return
normal return
```

The first 12_8 bytes will contain the name (with trailing nulls, if necessary); byte 13 will contain a null terminator.

Possible errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read or write into system area
74	ERMPR	Address outside address space.

File Maintenance Commands

The commands in this section relate to individual files; they enable you to create, delete, set position, and check the status of files. The file maintenance commands are:

.CCONT	Create a contiguous file with data words zeroed.
.CONN	Create a contiguous file with *no* data words zeroed.
.CRAND	Create a random file.
.CREA	Create a sequential file.
.DELET	Delete a file.
.RENAM	Rename a file.
.GPOS	Get the current file pointer.
.SPOS	Set the current file pointer.
.STAT	Get a file's status.
.RSTAT	Get a link entry's resolution file status.
.CHSTS	Get a channel's file information.
.UPDAT	Update an open file's size information.

Each file maintenance command requires you to specify the file name(s) by means of a byte pointer to the file name. In the byte pointer, bits 0-14 contain the word address which holds or will receive the first byte. Bit 15 indicates which half: 0 is left, 1 is right.

If you want to specify an extension, separate it from the filename with a period (.). For example, the word at location BTPR contains a byte pointer to a properly specified file name, MYFILE.SR.

```
        .TXTM 1
        .
        .
        .
BPTR: .+1*2
        .TXT "MYFILE.SR"
```

File names can include directory specifiers.

If you attempt to create a file with the same name as a device in the current system (e.g., SLPT), the system will treat the command as a no-op and take the normal return.

Create a Contiguously-Organized File with All Data Words Zeroed (.CCONT)

This call creates a contiguously-organized file with all data words initialized to zero. If the file's name exists as a link entry, and if no resolution file exists for this link entry, RDOS will create a contiguous resolution file.

Required input

AC0 - Byte pointer to the file name.

AC1 - Number of disk blocks in the file.

Format

```
.SYSTM
.CCONT
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	File already exists.
27	ERSPC	Insufficient disk space to create a SYS.DR entry for this file.
46	ERICB	Insufficient number of free contiguous disk blocks available to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
124	ERZCB	Attempt to create a zero length contiguous file.

Create a Contiguously-Organized File with No Zeroing of Data Words (.CONN)

.CONN creates a contiguously-organized file; it is faster than .CCONT because RDOS doesn't need to zero the data words. If the file's name exists as a link entry, and if no resolution file exists for this link entry, RDOS will create a contiguous resolution file.

Required input

AC0 - Byte pointer to filename.

AC1 - Number of disk blocks in the file.

Format

```
.SYSTM
.CONN
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	File already exists.
27	ERSPC	Insufficient disk space to create a SYS.DR entry for this file.
46	ERICB	Insufficient number of free contiguous disk blocks available to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
124	ERCZB	Attempt to create a zero length contiguous file.

Create a Randomly-Organized File (.CRAND)

This command makes an entry for the file name of a randomly-organized file in the system file directory (SYS.DR), and assigns the first index block to the file. If the file's name exists as a link entry, and if no resolution file exists, RDOS will create a random resolution file.

Required input

AC0 - Byte pointer to the file name.

Format

```
.SYSTM
.CRAND
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	File already exists.
27	ERSPC	Insufficient disk space to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address.
100	ERMDE	Error detected in MAP.DR of non- master device.
101	ERDTO	Disk timeout occurred.

Create a Sequentially Organized File (.CREAT)

This call creates an entry in the system file directory (SYS.DR) for the file name of a sequentially-organized file, and assigns the first file block. If the file's name exists as a link entry, and if no resolution file exists for this link entry, RDOS will create a sequential resolution file.

Required input

AC0 - Byte pointer to the file name

Format

```
.SYSTM
.CREAT
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	File already exists.
27	ERSPC	Insufficient disk space to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Delete a File (.DELET)

Use this command to delete a file and its entry in the system file directory. Do *not* delete link entry names with this call. If you attempt to delete a link entry name, its *resolution file* will be deleted unless 1) either the link access or resolution entry attributes words contain the permanent attribute (in which case RDOS returns error ERDE1), or 2) a resolution file doesn't exist (ERDLE returned).

Required input

AC0 - Byte pointer to filename.

Format

```
.SYSTM
.DELET
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
13	ERDE1	File is permanent.
53	ERDSN	Directory specifier unknown.
56	ERDIU	Directory in use.
57	ERLDE	Link depth exceeded.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
100	ERMDE	Error detected in MAP.DR of nonmaster device.
101	ERDTO	Disk timeout occurred.
102	ERENA	Link access not allowed (N attribute).

Rename a File (.RENAM)

This call renames a file. You may rename a file in a different directory, as long as you use the same directory specifier in both the current name and new name.

Required input

AC0 - Byte pointer to the current filename

AC1 - Byte pointer to the new name.

Format

```
.SYSTM
.RENAM
error return
normal return
```

After a normal return, the old name no longer exists in the file directory.

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Attempt to create an existent name. (AC1)
12	ERDLE	Attempt to rename a nonexistent file. (AC0)
13	ERDE1	Attempt to rename a permanent file. (AC0)
35	ERDIR	Files specified in different directories.
53	ERDSN	Directory specifier unknown.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Get a File's Current Directory Status (.STAT/.RSTAT)

Use either of these system calls to get a copy of the current directory. status information for a file. These calls write a copy of the 22_8 word UFD (as it exists on disk) into the area you specify.

You can then access this information via the indicated displacements defined below. If the file is open, the information returned is a snapshot of the UFD as it existed on disk at the time of the most recent .CLOSE or .UPDAT.

Use system call .STAT to return the UFD of a file. Use .RSTAT on links to find the UFD of the link's rèsolution file. .RSTAT and .STAT have the same effect on a nonlink file.

Following is a template of a file UFD with displacement mnemonics:

Offset or Displacement	Mnemonic	Content
00000-000004	UFTFN	File name (ASCII file number for open tape file).
000005	UFTEX	Extension.
000006	UFTAT	File attributes.
000007	UFTLK	Link access attributes.
000010	UFTBK	Number of the last block in the file (i.e., block count -1).
000011	UFTBC	Number of bytes in the last block.
000012	UFTAD	Starting logical block address of the file (the random file index for random files).
000013	UFTAC	Year/day last accessed.
000014	UFTYD	Year/day created, update or closed after write.
000015	UFTHM	Hour and minute the file was created, updated, or closed after write.
000016	UFTP1	UFD temporary.
000017	UFTP2	Number of data words on a disk block.
000020	UFTUC	User count (1B0 =.EOPEN or .APPEND or .TOPEN; 1B1 = .OPEN)
000021	UFTDL	DCT link. Bits 10-16 contain device code of device which holds file; left byte is unused, except for large disks, for which bits 0-2 contains the high order of the disk address.

If you issue .STAT to a link entry, RDOS returns the link's UFD. In a link UFD, words 7 and 14_8 have mnemonics UFLAD and UFLAN; words 7-13 and 14-21 contain the link's alternate directory specifier (if any) and an alias (if any), respectively.

Required input

AC0 - Byte pointer to file name string.

AC1 - Starting address of 22_8 word UFD data area.

Format

```
.SYSTM                          .SYSTM
.STAT              or           .RSTAT
error return                    error return
normal return                   normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
33	ERRD	Attempt to read or write into system file space.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded (.RSTAT only).
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Device timeout.

Get the File Directory Information for a Channel (.CHSTS)

.CHSTS returns a copy of current directory status information for whatever file is currently open on a specified channel. RDOS returns directory status information as a copy of the 22$_8$ word UFD, as described in .STAT, except that it shows file status as of last file I/O (by the system, not by you) of this channel. For example, .CHSTS would return the status after a .WRL, whereas .STAT/.RSTAT would show status, on disk, as of the last update or close.

Required input

AC0 - Starting address of data area. This area must be at least 22$_8$ words long.

Format

```
.SYSTM
.CHSTS n            ;n is the file's channel number
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	No file opened on the given channel.
33	ERRD	Attempt to read into system area.
75	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Update the Current File Size (.UPDAT)

This call allows you to update the size information in a file's UFD while the file is open. The UFD contains a file's size, creation date, attributes, and other information. Specifically, this call updates information in UFTBK and UFTBC in the disk UFD for the file opened on a specified channel, and it writes all modified system buffers which are not in use to ensure that the file contains all information that your program has written into it.

This call is particularly useful when a file is open for a long time. Any file that is open during a system failure may have inaccurate size information in its UFD; if so you will be unable to read new data. By .UPDATing the file frequently, you keep its UFD current and minimize the amount of data which could be lost.

Format

```
.SYSTM
.UPDAT n            ;n is the file's channel number
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	File not opened.
101	ERDTO	Disk timeout occurred.

File Attribute Commands

File attribute commands allow you to check or change the current attributes of a file; you can also use them to check device characteristics. The bit settings of AC0 determine the file attributes; AC1 contains the device characteristics of the file.

This section describes the following calls:

.CHATR	Change the attributes of the file opened on channel n
.GTATR	Get the attributes or characteristics of the file opened on channel n.

Note that these calls work only on an open file. For link commands, see the next section.

Change File Attributes (.CHATR)

This command changes the access attributes of an open file (or the resolution entry attributes, as viewed from a link entry), according to the contents of AC0.

When you create a file, it has no attributes. If a link user or a user who has opened via .ROPEN issues .CHATR. RDOS temporarily changes his/her copy of the file attributes until he/she closes the file; however, the true resolution entry attributes persist. You must open a file (OPEN or .OPEN) before you can change its attributes.

Note that RDOS provides two special attribute bits; you can use these to define your own unique file access specifications.

Format

```
.SYSTM
.CHATR n            ;n is the file's channel number
error return
nᴖrmal return
```

Required input

AC0 - an attribute word which contains bits set according to the attributes you want. Set the contents of AC0 according to the following bit/attribute relationships:

Bit	Symbolic Attribute	Mnemonic	Meaning
1B0	R	ATRP	Read-protected file; cannot be read.
1B1	A	ATCHA	Attribute-protected file. No attribute can ever be changed after you set this bit.
1B2	S	ATSAV	Save file (core image file).
1B7	N	ATNRS	No link resolution allowed.
1B9	?	ATUS1	First user-definable attribute for the file.
1B10	&	ATUS2	Second user-definable attribute for the file.
1B14	P	ATPER	Permanent file; cannot be deleted or renamed.
1B15	W	ATWP	Write-protected; cannot be written.

The following are disk file characteristics, RDOS assigns them when you create a file; you cannot change them.

Bit	Characteristic	Mnemonic	Meaning
1B3	L	ATLNK	Link entry.
1B4	T	ATPAR	Disk partition.
1B5	Y	ATDIR	Subdirectory.
1B6	-	ATRES	Link resolution file (temporary). Other file attributes persist for the duration of the open.
1B12	C	ATCON	Contiguous file.
1B13	D	ATRAN	Random file.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
14	ERCHA	Illegal attempt to change file attributes (file has A attribute).
15	ERFOP	No file open on this channel.
101	ERDTO	Disk timeout occurred.

Get the File Attributes and Characteristics (.GTATR)

Use this command to obtain the attributes or device characteristics of a file.

Format

```
.SYSTM
.GTATR n            ;n is the file's channel number
error return
normal return
```

When RDOS returns, AC0 will contain the file attributes. See the .CHATR command for a description of the bit positions that specify attributes. AC1 will contain the device characteristics of the file. These pertain to files on reserved devices, e.g., SLPT. These *do not* reflect the characteristic disable mask supplied when the file was opened. Use this bit/characteristics table to interpret the bit configuration returned in AC1:

Bit	Mnemonic	Meaning
1B0	DCSPC	When the file is a spoolable device, 1B0 means spooling enabled (disabled if 0B0).
1B0	DCDIO	When file is an MCA link, this means that protocol is suspended on transmit.

Bit	Mnemonic	Meaning
1B1	DCC80	80-column device.
1B2	DCLTU	Device changes lower case ASCII to upper case.
1B3	DCFFO	Device requiring form feeds on opening.
1B4	DCFWD	Full word device (reads or writes more than a byte.)
1B5	DCSPO	Spoolable device.
1B6	DCLAC	Output device requiring line feeds after carriage returns.
1B7	DCPCK	Input device requiring a parity check; output device requiring parity to be computed.
1B8	DCRAT	Output device requiring a rubout after every tab.
1B9	DCNAF	Output device requiring nulls after every form feed.
1B10	DCKEY	CTRL Z end-of-file, backslash line delete, and rubout character delete are disabled for this keyboard input device.
1B11	DCTO	Teletypewriter output device or equal leader and trailer for STTP and SPTP.
1B12	DCCNF	Output device without form feed hardware.
1B12	DCLCD	Input device is 6053-type terminal.
1B13	DCIDI	Input device requiring operator intervention.
1B14	DCCGN	Output device without tabbing hardware.
1B15	DCCPO	When file is STTR/STTP: output device requiring leader and trailer.
1B15	DCSTO	When file is MCA line: User-Specified MCA transmitter timeout.
1B15	DCNI	When file is MUX line: no CTRL-A or CTRL-C interrupts from this line.
1B15	DCSTB	When file is SCDR: trailing blanks are suppressed.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to get attributes of an unopened file.
101	ERDTO	Disk timeout occurred.

Link Commands

As we described in Chapter 2, RDOS permits you to link files in one directory to files in other directories. Either directory can be a primary partition, secondary partition, or subdirectory. The link commands are:

.LINK	Create a link entry.
.UNLK	Delete a link entry.
.CHLAT	Change the link access attributes of a file.

Create a Link Entry (.LINK)

This call creates a link entry in the current directory to a file in the same or another directory. This link entry may or may not have the same name as the resolution file; if not, the link entry name is an alias. No attributes restrict a link when you create it, but it cannot reach the resolution file without satisfying both the link entry and the file access attributes of the resolution entry. Your program can alter the link access rights (but not the file access rights) of any nonlink file by using the .CHLAT call.

Typical examples of alternate directory/alias name strings are as follows:

Linkname	Resolution Filename	Meaning to RDOS
LFE.SV	LFE.SV	Create link entry LFE.SV in the current directory; link it to resolution file LFE.SV on the current directory's parent partition.
LFE.SV	SAM:LFE.SV	Create link LFE.SV in the current directory: link it to resolution file LFE.SV in directory SAM.
NLFE.SV	DP1:LFE.SV	Create link NLFE.SV in the current directory; link it to resolution file LFE.SV in primary partition DP1.

Required input

AC0 - Byte pointer to link entry name string.

AC1 - Zero if the link and resolution file have same name, and if the resolution file is in the parent partition. Byte pointer to the name string if the link entry has an alias name, or is not on the parent partition. You can omit a directory specifier from the resolution file name if the resolution file is on the link entry's parent partition. For example, in Figure 2-4, SECONDPART is SUBDIR's parent partition; Dxn is SECONDPART's parent directory.

Format

```
.SYSTM
.LINK
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Link entry name already exists.
27	ERSPC	Insufficient disk space to create SYS.DR entry.
53	ERDSN	Directory specifier unknown.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Delete a Link Entry (.ULNK)

This call deletes a link entry (created earlier by LINK or .LINK) in the directory to which the link entry name points. This call does not delete other links of the same name in other directories. You must be sure that the link entry you are deleting does not also exist between other links and the resolution entry; if it does, you will not be able to resolve these more remote links after this deletion.

Required input

AC0 - Byte pointer to the link entry name string.

Format

```
.SYSTM
.ULNK
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
53	ERDSN	Directory specifier unknown.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space
75	ERNLE	Not a link entry.
101	ERDTO	Disk timeout occurred.

Change Link Access Entry Attributes (.CHLAT)

This command changes the link attributes word of the file opened on a channel, according to the contents of AC0. When you open a file via a link entry, the attributes you see will be a composite of the resolution entry's file attributes and your copy of the link access entry attributes. When you create a file, no link entry access attributes exist.

Note that RDOS provides two special attribute bits; you can use them to define your own unique link access specification.

Required input

AC0 - File attributes word (identical to .CHATR)

Format

```
.SYSTM
.CHLAT n          ;n is the channel number
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
14	ERCHA	Resolution entry is attribute-protected (has A attribute).
15	ERFOP	No file is open on this channel.
101	ERDTO	Disk timeout occurred.

Input/Output Commands

This section describes the calls your program can use to write data to, and read data from, an existing, open file. It begins by describing the five I/O modes available, and proceeds to explain the calls which open and close a file.

It then covers the calls you can use to change position in a file, and finally lists the different writing/reading calls themselves.

Generally, you can do nothing with a file until you have opened it and given it a channel number with one of the .OPEN commands: .OPEN, .EOPEN, .ROPEN, .APPEND, or .MTOPD.

Remember that a file can be a device (e.g., $TTI, console input; or QTY:xx, multiplexor line) or a disk file (e.g., MYFILE.SR), which can include a directory specifier (e.g., DP1:MYFILE.SR) if you have initialized the directory.

These are the file I/O calls:

.OPEN n	Open a file for I/O on channel n.
.EOPEN n	Open a file for exclusive writing on channel n.
.ROPEN n	Open a file for reading only on channel n.
.APPEND n	Open a file for appending on channel n.
.GCHN	Get the number of a free channel.
.CLOSE n	Close the file on channel n.
.RESET	Close all files.
.GPOS n	Get the position of the file pointer.
.SPOS n	Set the position of the file pointer.
.RDL n	Read an ASCII line from a file.
.WRL n	Write an ASCII line to a file.
.RDS n	Read sequential bytes from a file.
.WRS n	Write sequential bytes to a file.
.RDR n	Read a 64-word record.
.WRR n	Write a 64-word record.
.RDB n	Read (or Write) a series of disk blocks from or to
.WRB n	a file, without a system buffer.
.MTOPD n	Open a mag tape or cassette file for free-form I/O.
.MTDIO n	Write or read data to or from a mag tape or cassette file in free form.

If RDOS detects an error when it executes your I/O command, it will retry the command (if possible) before reporting the error with code ERFIL.

RDOS provides five basic modes for reading and writing files:

- line
- sequential
- random record
- direct block
- free form (tape)

This section presents the calls for these modes in order.

You will generally use *line* and *sequential* * mode for ASCII character strings and binary files, respectively. *Random record* mode allows you to read or write 64-word records. *Direct-block* I/O allows you to transfer a contiguous group of disk blocks without a system buffer. *Free form* I/O allows you to read or write free form blocks of data to mag tape.

*The RDOS System Library contains a module to speed up line and sequential mode operations. This module is called the *Buffered I/O Package*, and is described in that Application Note.

In *line* mode, the system assumes that the data you want to read or write consists of ASCII character strings, terminated by either a carriage return, a form feed, or a null character. RDOS processes file data line-by-line in sequence from the beginning of the file to its end.

In line mode, the system handles all device-dependent editing at the device driver level. For example, it ignores line feeds on paper tape input devices and supplies them after carriage returns to all paper tape output devices. Furthermore, reading and writing never require byte counts, since reading continues until RDOS reads a terminator and writing proceeds until you write a terminator. The line mode commands are Read a Line (.RDL) and Write a Line (.WRL).

The second mode is the unedited *sequential* mode. In this mode, RDOS transmits data exactly as it reads it from or writes it to the file or device. You must use this mode for processing sequential binary files. To use sequential mode, your program must specify the byte count necessary to satisfy your read or write request. The sequential mode commands are Read Sequential (.RDS) and Write Sequential (.WRS).

In line or sequential modes, your position within a file is always the position at the end of your last line or sequential mode call, or .SPOS call. The first read or write occurs at the beginning of the file, unless your program opened the file for appending.

The third mode, *random record*, permits random access to fixed-length records within random or contiguous disk files. The fixed length of a random record is 100_8 words. The random calls are .RDR and .WRR.

The fourth mode, *direct block I/O*, allows you to transfer a continuous group of blocks in a random or contiguous file without using a system buffer. RDOS uses sequential memory locations in the transfer, and it transfers only 512-byte blocks of data between memory and disk. You can transfer only an unbroken series of relative block numbers; i.e., you may process the third, fourth, and fifth blocks in a file in a single call, but not the third, fifth, and sixth blocks. You can execute direct-block I/O with .RDB and .WRB.

If you have a mapped system, you can employ window mapping, which permits *extended direct-block I/O*. In this mode, your program can transfer disk blocks to and from extended address space via .ERDB and .EWRB, as described in Chapter 4.

Finally, *free form* I/O permits you to read or write free form blocks of data to magnetic tape. With free form I/O, you can read or write from two to 4096- word data records, you can space forward or backward through one to 4096 data records or to the start of a new data file, and you can read the transport status word. To use free form I/O, you must open a file via .MTOPD, and direct its operation via .MTDIO. You cannot mix .MTDIO with .WRL, or .WRS on the same tape drive.

Open a File (.OPEN)

Before your program can issue other I/O commands, it must associate a file to an RDOS channel number. .OPEN associates a file with a channel number and makes the file available to anyone for both reading and writing. The .OPEN command does not guarantee exclusive use of the file; other users may also have opened the file via .OPEN and modified its contents. Everyone using a file must close it before anyone can delete or rename it. In RDOS, there is no command to reduce the size of a file. This means that files never shrink, and they maintain space for all material written to them by any user. If you want to remove redundant or useless material from a file, you can either edit it with a text editor utility, or you can overwrite the useless data with nulls or new material, using file position and system write calls.

Required input

AC0 - Byte pointer to the filename

AC1 - Characteristic disable mask (except for MCA lines; see below). For every bit you set in the mask word, RDOS disables the corresponding device characteristic for the duration of the .OPEN. (See .GTATR in the File Attributes Commands section of this chapter.)

For example, if you want to read an ASCII tape without parity checking from the paper tape reader, you can disable checking by the following:

```
           LDA 0,READR
           LDA 1,MASK
           .SYSTM
           .OPEN 3
           .
           .
           .
READR:     .+1*2
           .TXT "SPTR"
MASK:      DCPCK        ;DISABLE PARITY
                        ;CHECKING.
```

RDOS normally restricts console output to 80 columns. If your terminal is a DASHER, you can instruct RDOS to print the full 132 columns by opening $TTO ($TTO1) with disable bit DCC80 set; e.g.,

```
          .
          LDA 0, NTTO
          LDA 1, DMASK
          .SYSTM
          .OPEN n
          .
          .
    NTTO:.+1*2
          .TXT "$TTO"
    DMASK: DCC80
.
```

To use system mnemonics like mask and error words, you should assemble your program with a macroassembler, and the assembler's symbol table file must include PARU.SR. See Chapter 1, *System Library and Source Files*, for more information on this.

In general, you will want to preserve all device characteristics defined by the system. To preserve them, insert a SUB 1, 1 instruction before the .OPEN call.

Format

```
.SYSTM
.OPEN n              ;n becomes the channel
                     ;number of the file
                     ;until n is closed.
error return
normal return
```

Note that RDOS will interleave line printer output if multiple tasks in the same program write to the printer.

To open an MCA line for transmit, you must specify a *transmit timeout period* (not a mask) in AC1. Set AC1 to 0 to specify the default timeout period (655 seconds); for a shorter timeout period, set AC1 to 1 (specify the actual timeout period in the write-sequential call, .WRS).

To open an MCA line for receiving, pass 0 in AC1.

If the file opened requires leader, RDOS will output it on the .OPEN. If the file opened requires intervention, RDOS will display the message LOAD *filename*, STRIKE ANY KEY.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
27	ERSPC	File space exhausted.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File opened for exclusive use (.EOPEN).
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempted open of an open tape file.

Open a File for Exclusive Write Access (.EOPEN)

This command gives you exclusive write access to a file. Thus only you can modify a given file when you open it via .EOPEN, although other users may gain read access to this file via .ROPEN.

Required input

AC0 - Byte pointer to file name.

AC1 - Characteristic disable mask.

Format

```
.SYSTM
.EOPEN n          ;n is the file's channel number
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File already opened for writing.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempt to open a file which is already open.

Open a File for Reading Only (.ROPEN)

This call opens a file for reading only. Your program can gain read-only access to a file which is currently open by either .EOPEN, .OPEN, or another .ROPEN. Thus several users may access a file for reading only while one of those users has write access privileges to the file. All users must have closed the file before anyone can delete or rename it.

Required input

AC0 - Byte pointer to file name

AC1 - Characteristic disable mask

Format

```
.SYSTM
.ROPEN n          ;n is the file's channel number
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempt to open an open tape file.

Open a File for Appending (.APPEND)

.APPEND is identical to .EOPEN, except that it opens a file specifically for appending.

If your program tries to *read* a file which you have opened for appending, RDOS will return error code EREOF (end-of-file), because the file pointer is positioned after the last byte.

In a BATCH environment, if you want your program to output to SYSOUT, you must open SYSOUT for appending (not simply opening).

Required input

AC0 - Byte pointer to the filename

AC1 - Device characteristic disable mask

Format

```
.SYSTM
.APPEND n          ;n is the file's channel number
error return
normal return
```

On a disk, RDOS opens the file, and appends whatever you write to that file. On a magnetic tape device, RDOS opens the tape file and reads to the end-of-file (EOF); it then writes from that point. On a line printer, RDOS opens the printer without a form feed.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
3	ERICD	Illegal command for device.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempt to open a file that is already open.

Open a Magnetic Tape or Cassette Unit for Free Format I/O

The commands .OPEN, .EOPEN, .ROPEN and .APPEND cannot open a tape file for free format I/O. See .MTOPD and .MTDIO at the end of Input/Output Commands, this chapter.

Get the Number of a Free Channel (.GCHN)

This call returns (in AC2) the number of a free channel. Your program can then use AC2 to open a file via one of the open file calls. .GCHN does not open a file on a free channel; it merely indicates a channel that is free at the moment. Occasionally, in a multitask environment, you will find that the channel .GCHN indicated is no longer free when you issue your open. If this happens, you will receive error return ERUFT; reissue the call .GCHN, to discover another free channel.

Format

```
.SYSTM
.GCHN
error return
normal return
```

Upon a normal return, RDOS returns the free channel number in AC2.

Possible error

AC2	Mnemonic	Meaning
21	ERUFT	No channels are free.

Close a File (.CLOSE)

You must close a file after use to update its UFD directory information, or delete it, or release its directory or device. When you close a file, its channel number becomes available for other I/O. The calls .RTN, ERTN, .BREAK or .RESET automatically close all channels.

Format

```
.SYSTM
.CLOSE n          ;Close channel n
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to close a channel not in use.
101	ERDTO	Disk timeout occurred.

Close All Files (.RESET)

This command closes all open files after writing any partially-filled system buffers. You can issue .RESET in a multitask environment only when no other task is using a channel.

Format

```
.SYSTM
.RESET
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
101	ERDTO	Disk timeout occurred.

Get the Current File Pointer (.GPOS)

Use this call to determine the next character position within a file where program writes or reads will occur. RDOS indicates a relative character position within a file by a double-precision byte pointer. This is a two-word byte pointer containing the high-order portion of the byte address in AC0 and the low-order portion of the byte address in AC1. Bit 15 of the second word indicates the byte selection (left or right), as shown in Figure 3-1.

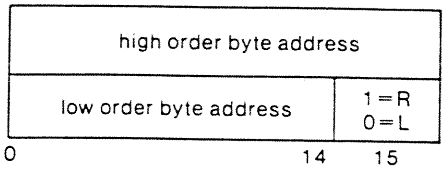


Figure 3-1. Double-Precision Byte Pointer

Format

```
.SYSTM
.GPOS n          ;n is the file's channel number
error return
normal return
```

RDOS returns the pointer position in AC0 and AC1, as described above. RDOS returns zero if you open a nondisk file on channel n.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	No file is open on this channel.

Set the Current File Pointer (.SPOS)

This call sets the current system file pointer to a new character position for future program file writes or reads. RDOS indicates the relative character position within a file by the double-precision byte-pointer described above in .GPOS. For a mag tape or cassette, you can specify only position 0 (the file starting location).

This call enables you to access characters and lines randomly within any block of a given file. You can read a character after writing or rewriting it simply by backing up the pointer to its previous position.

If you set the file pointer beyond the end of the file, RDOS automatically extends the length of the file. If the file is contiguous - hence cannot be extended - RDOS will take the error return, and pass ERSCP in AC2.

Required input

AC0 - High-order portion of byte pointer.

AC1 - Low-order portion of byte pointer.

Format

```
.SYSTM
.SPOS n          ;n is the file's channel number
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to reference an unopened file.
64	ERSCP	File position error.

Read a Line (.RDL)

.RDL reads an ASCII line from a file to your specified area. AC0 must contain a byte pointer to the starting byte address within user memory into which RDOS will read the line. This area should be 133_{10} bytes long.

Reading terminates normally after RDOS has read either a carriage return, form feed, or null, and transmitted it to your program. The system stops reading and takes the error return if it transmits 133 characters without detecting a carriage return, form feed, or null, or upon detection of a parity error, or end-of-file.

If RDOS is reading from a multiplexed line, it also terminates reading if it reads a pre-assigned interrupt character. See *Multiplexors* in Chapter 2.

If the file you are reading from is the keyboard (STTI, STTI1), keyboard controls work as usual (unless you have masked DCKEY in AC1-- see .GTATR). Rubout deletes the preceding character, and backslash (SHIFT-L) deletes the preceding line, from the keyboard stream. RDOS echoes all printing characters and ignores line feeds. You can indicate an end of file by pressing CTRL-Z. Note that when you are reading from a multiplexed line, ESC also indicates an end of file.

RDOS will always return the number of bytes read (including the carriage return, form feed, or null) in AC1. If the read terminates because of a parity error, RDOS stores the character having incorrect parity as the last character read and clears the parity bit. You can always compute the byte pointer to the bad character as (AC0) + (AC1)-1. (Note: (AC0) means the contents of AC0.)

Required input

AC0 - Byte pointer to receiving buffer.

Format

```
.SYSTM
.RDL n           ;Read from channel n
error return
normal return
```

After a normal return, AC1 will contain the number of bytes read.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for devce.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	Attempt to reference a file not open.
22	ERLLI	Line limit (133 nonterminator characters) exceeded.
24	ERPAR	Parity error (tape - possibly dirty heads).
30	ERFIL	File read error (bad tape; possibly dirty heads).
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous reads from the same multiplexor (ALM/QTY) line.
74	ERMPR	Address outside address space (mapped only).
101	ERDTO	Disk timeout occurred.
106	ERCLO	Channel closed by another task.

Write a Line (.WRL)

.WRL is the counterpart of .RDL; it writes an ASCII line to the file open on the specified channel. AC0 must contain a byte pointer to the starting byte address within user memory from which characters will be written.

If you have opened the file with .OPEN or .EOPEN, writing starts at the beginning of the file (unless you have moved the file pointer via the .SPOS command). Writing begins at the end of the file if you open it via .APPEND. Normally, the system stops writing when it detects a null, a carriage return, or a form feed. Abnormally, it stops writing after transmitting 132 (decimal) characters without a carriage return, a null, or a form feed as the 133rd character.

Upon termination, AC1 contains the number of bytes written from your area to the file. The null terminator does not force a carriage return or line feed. A carriage return generates a line feed upon output (if the device characteristics so dictate).

Required input

AC0 - Byte pointer to starting byte address.

Format

```
.SYSTM
.WRL n              ;Write to the file on channel n
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End-of-file when writing to a contiguous file.
10	ERWPR	Attempt to write to a write-protected file.
15	ERFOP	Attempt to write a file not opened.
22	ERLLI	Line limit (132 characters).
27	ERSPC	Out of disk space.
34	ERDIO	File accessible by direct-block I/O only.
47	ERSIM	Simultaneous writes to the same multiplexor (ALM/QTY) line.
74	ERMPR	Address outside address space
101	ERDTO	Disk timeout occurred.
103	ERMCA	The MCA receiver on this channel issued no transmit request.
104	ERSRR	MCA transmission terminated by receiver (short receive request).
106	ERCLO	Channel closed by another task.

Use of the Card Reader ($CDR) in .RDL and .RDS Commands

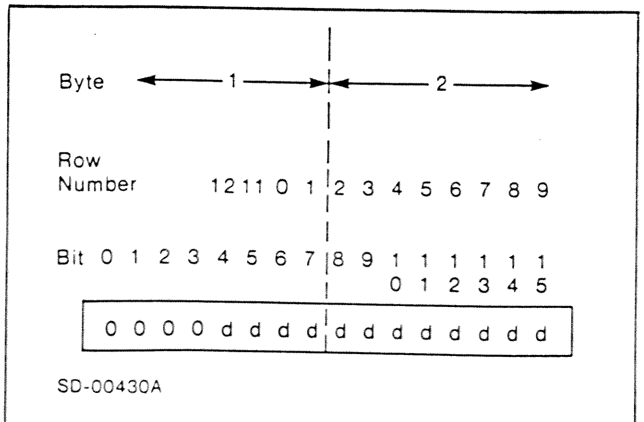
When you use the card reader as an input device to .RDL, indicate an end-of-file by punching all rows in column 1 (multipunch the characters "+", "-", and 0 through 9). Hollerith-to-ASCII translation occurs on a .RDL, not on a .RDS. The translation assumes the keypunch codes shown in Appendix B.

A .RDL terminates upon the first trailing blank unless your .OPEN command suppressed DCSTB, thus causing RDOS to transfer all 80 characters. If RDOS transfers all 80 characters, it will append a carriage return as the eighty-first character, unless your .OPEN command suppressed DCC80 (allowing RDOS to process a maximum of 72 characters). The system replaces each illegal character with a backslash.

In .RDL calls, RDOS ignores all columns following the EOF. The card reader driver permits an unlimited amount of time to elapse until it reads the next card, thus permitting the operator to correct pick errors or insert new card files. The card reader driver employs double buffering, and you will lose at least one card image if you close prematurely; therefore your program must wait until RDOS reads the last card or end-of-file to close $CDR.

You can close the reader after it has read an end-of-file card, reopen it without losing any data, and continue card reading. When RDOS reads an end-of-file card it returns a byte count of 0 and error code EREOF. If you issue another .RDL, it will read the next card normally.

If you issue .RDS (see below), RDOS reads the card in image binary. It uses each two bytes to read a single column, packing them as shown in Figure 3-2.



Byte
1
2
Row Number
12 11 0 1 2 3 4 5 6 7 8 9
Bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 0 0 0 d d d d d d d d d d d d
SD-00430A

Figure 3-2. Image Binary Card Reading

Each "d" will be 1 for every punched hole in the column. In .RDS, you signify an end-of-card (EOC) by a byte pair containing the word 100000. Thus, to read two entire 80-column cards, one card at a time, you would issue two successive .RDS calls for 162 bytes each. If you requested only 160 bytes for each read, the second .RDS would return the first end-of-card word, and the first 79 columns of the second card.

Read Sequential (.RDS)

In the sequential mode, RDOS transmits data exactly as it reads it to or writes it from a file. You must use this mode for binary data, and it is often useful for MCA transmissions.

The .RDS call tells RDOS to read data exactly as it is in the file, unless it is reading from the system console. When reading sequentially from a system console, RDOS sets the parity bits to zero. Note that RDOS does not recognize CTRL Z from the console as an end-of-file character in this mode. Upon detection of an end-of-file, RDOS will return the partial bytecount in AC1.

Required input

AC0 - Byte pointer to the starting byte address within user memory into which RDOS will read the data.

AC1 - Number of bytes to be read.

Format

```
.SYSTM
.RDS n              ;Read from channel n
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End of file.
7	ERRPR	Atempt to read a read-protected file.
15	ERFOP	Attempt to reference a file not open.
24	ERPAR	Parity error (tape). Often caused by dirty heads.
30	ERFIL	File read error (bad tape or dirty tape heads).
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous reads from same multiplexed line.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
103	ERMCA	The MCA transmitter issued no transmit request.
106	ERCLO	Channel closed by another task.

Write Sequential (.WRS)

.WRS is the counterpart of .RDS; it writes data verbatim from memory to a file. Note that RDOS recognizes no character as an end-of-file in this mode.

If you open a file via .OPEN or .EOPEN, RDOS starts writing at the beginning of the file (unless you moved the file pointer by the .SPOS command after opening). If you opened the file via .APPEND, RDOS starts writing at the end of the file.

Required input

AC0 - Byte pointer to the starting address of the data within user memory.

AC1 - Number of bytes to be written.

Format

```
.SYSTM
.WRS n              ;Write to channel n
error return
normal return
```

To transmit (write) data over an MCA line, you must pass an even byte pointer in AC0, and specify an even byte count in AC1. If you .OPENed this MCA channel and specified a nondefault retry period in AC1, then you must specify the length of the timeout period in the left byte of AC2. Each retry takes about 200 milliseconds, and acceptable values input in AC2 are 1 to 377_8. If the left byte of AC2 is 0, RDOS will allot the maximum transmit retry period (about 655 seconds).

To send an end-of-file over an MCA line, set AC1 to 0; RDOS will disregard the contents of AC0. For more on MCA programming, see Chapter 8.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel numer.
3	ERICD	Illegal command for device.
6	EREOF	End-of-file when writing to a contiguous file.
10	ERWPR	Attempt to write a write-protected file.
15	ERFOP	Attempt to write a file not open.
27	ERSPC	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous writes to the same QTY/ ALM line.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
103	ERMCA	The MCA receiver on this channel issued no receive request.
104	ERSRR	MCA transmission terminated by receiver (short receive request).
106	ERCLO	Channel closed by another task.
113	ERNMC	No outstanding receive request.

Read (or Write) Random Record (.RDR or .WRR)

These calls allow your program to read (or write) one 64-word record in either a random or contiguous disk file. There are four 64-word records in a disk block; for the first disk block in a file, these are numbered 0, 1, 2, and 3. For the second block, the numbers are 4, 5, 6, 7, and so on. You need consider record numbers only for the random record calls (to read or write blocks (i.e., four records at a time), you'd use .RDB/.WRB; to read or write lines you'd use the read/write line (.RDL) or sequential (.RDS/.WRS) calls.

Required input

AC0 - Destination memory address.

AC1 - Record number (record numbers start with 0).

Format

```
.SYSTM
.RDR n          ;Read from the file
error return    ;opened on channel n.
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	Attempt to read past the end of a contiguous file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	No file is open on this channel.
30	ERFIL	File read errors (mag tape or cassette - probably a bad tape).
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct-block I/O only.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Write Random Record (.WRR)

.WRR writes a 64-word record from memory to a randomly- or contiguously- organized disk file. RDOS will write 64 words to the record number you specify, starting from the address you pass in AC0.

Required input

AC0 - Memory address.

AC1 - Destination record number.

Format

```
.SYSTM
.WRR n          ;Write to the file
error return    ;opened on channel n.
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	Attempt to write past the end of a contiguous file.
10	ERWPR	Attempt to write a write-protected file.
15	ERFOP	Attempt to reference a file not opened.
27	ERSPC	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Read (or Write) a Series of Disk File Blocks (.RDB/.WRB)

These are direct block I/O calls. Use these calls in your program to transfer blocks to or from random or contiguous files. RDOS uses no system buffers for the transfer.

Blocks in random and contiguous disk files have a fixed length of 256_{10} words; they are numbered sequentially from 0. A .RDB for the first block in a file would transfer the 64-word records numbered 1, 2, and 3, as described under .RDR, above.

Required input

AC0 - Starting memory address for the block transfer.

AC1 - Starting relative block number in the series to be transferred.

AC2 - The left half of AC2 must contain the number of blocks which you want RDOS to transfer. If you set the channel number to 77, the right half of AC2 must contain the channel number.

Format

```
.SYSTM
.RDB (.WRB) n       ;n is the channel number
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
4	ERSV1	Not a random or contiguous file.
6	EREOF*	End of file.
7	ERRPR	File is read-protected (.RDB).
10	ERWPR	File is write-protected (.WRB).
15	ERFOP	File is not open.
27	ERSPC*	Disk space is exhausted.
30	ERFIL	File read error, mag tape or cassette. Probably a bad tape or dirty head.
33	ERRD	Attempt to read into system area (.RDB).
40	EROVA	File not accessible by direct- block I/O.
74	ERMPR	Address outside address space (mapped only).
101	ERDTO	Disk timeout occurred.

Open a Tape Unit and File for Free Format I/O (.MTOPD)

Before you can read or write in free format on a magnetic tape, you must open the device and associate it with a channel. Use the .MTOPD command to do this. After you have finished with the drive, release it.

.MTOPD is a global call; after you use it, you can access all files on the specified device.

To position a free format tape to a specific file, pass the file name to .MTOPD in the form MTn:m.

Required input

AC0 - Byte pointer to the magnetic or cassette tape file specifier.

AC1 - Characteristic disable mask (see .GTATR).

Aside from the tape file specifier, these parameters are identical to those for .OPEN. If you want to know more about device characteristics, see .OPEN and .GTATR above.

Format

```
.SYSTM
.MTOPD n       ;n is the channel number
error return
normal return
```

*Upon detection of error EREOF or ERSPC, RDOS returns the code in the right byte of AC2; the left byte contains the partial read or write count.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
3	ERICD	Illegal command for device.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use a channel already in use.
27	ERSPC	File space exhausted.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
111	ERDOP	Attempted open of an open tape file.

Perform Free Format I/O (.MTDIO)

This command gives you a direct interface with magnetic tape units on a machine level. Using .MTDIO, you can read or write data in variable length records from 2 to 4096 words long, you can space forward or backward from 1 to 4096 data records or to the start of a new data file, or you can perform other similar machine-level operations.

Before you can read or write in free format on a tape unit, you must open the unit for free format I/O with the .MTOPD system command. For information about the hardware characteristics, see *Magnetic Tape*, in *Programmer's Reference Manual For Peripherals*.

Required input (to read device status word)

AC1 - Command word - bits 1-3 set, other bits 0.

AC2 - Channel number if *n* equals 77.

Required input (for other .MTDIO operations)

AC0 - Memory address for data transfer.

AC1 - Command word, subdivided into the following fields:

bit 0: set to 1 for even parity, 0 for odd parity.

bits 1-3: set to one of these seven command codes: 0 - read (words)* 1 - rewind the tape 3 - space forward (over records or over a file of any size up to 4096 records) 4 - space backward (over records or over a file of any size up to 4096 records) 5 - write (words) 6 - write end-of-file (parity: odd for 9 track, even for 7-track) 7 - read device status word

bits 4-15: word or record count. If 0 on a space forward (or backward) command, and the file is no more than 4096 record, RDOS positions the tape to the beginning of the next (or previous) file on the tape. If 0 on a read command, RDOS reads words until it encounters either an end-of-record or 4096 words. If 0 on a write command, the system will write 4096 words.

AC2 - channel number if *n* equals 77.

Format

```
.SYSTM
.MTDIO n          ;n is the channel number
error return
normal return
```

*When reading a 7-track tape with odd parity (i.e., a tape not written on an RDOS system), the controller will not detect the end-of-file; it will read the first word in the next record as 007417. Thus RDOS appends the first record of each file (after the first file) to the end-of-file of the previous file.

Upon a read status command, if RDOS detects no system error, control will go the normal return and AC2 will contain a device status word with one or more of the bits shown in Figure 3-3 set:

bit 0, error (bit 1, 3, 5, 6, 7, 8, 10, or 14)

bit 1, data late
bit 2, tape is rewinding
bit 3, illegal command

bit 4, high density if set to 1, otherwise, low density (always 1 for cassettes)
bit 5, parity error
bit 6, end-of-tape

bit 7, end-of-file
bit 8, tape is at load point
bit 9, 1 for 9-track, 0 for 7-track (always 1 for cassettes)

bit 10, bad tape (or write failure)
bit 11, send clock (0 for cassettes)
bit 12, first character (0 for cassettes)

bit 13, write-protected or write-locked
bit 14, odd character (0 for cassettes)
bit 15, unit ready

SD-00540A

Figure 3-3. .MTDIO Status Word Bits

When your program issues a read, write, space forward, or space backward command, the command word in AC1 contains the number of words written (or read) or the number of records spaced. A word or record count is returned upon a premature end-of-file.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device (i.e., improper open).
15	ERFOP	Attempt to reference a file not opened.
40	EROVA	File not accessible by free form I/O.
74	ERMPR	Address outside address space.

Figure 3-4 summarizes the possible returns by .MTDIO and the values returned in AC1 and AC2. On hardware errors, RDOS sets bit 0 of TSW (in AC2); on system errors it clears this bit.

As with regular magnetic tape I/O, the system will perform 10 read retries before taking the error return. For write errors, the system will perform the following sequence 10 times before taking the error return: backspace, erase a length of tape, and write.

COMMAND	RETURN	AC1	AC2
Any .MTDIO command with a system error detected	Error	Same as input	System error code
Rewind	Normal	Original input lost	Transport Status Word (TSW)
Rewind (tape at load point,etc.)	Error		
Read Status	Normal	Original input lost	TSW
Read Status	Error		TSW
Read, Write, Space forward, Space backward	Normal	Word or record count	TSW
Read, Write, Space forward, Space backward	Error (only after 10 retries in read/write)		
Write end-of-file	Error	Original input lost	TSW

SD-00431A

Figure 3-4. .MTDIO Values Returned

Console I/O Commands

To transfer single characters between your console and AC0, use commands .GCHAR and .PCHAR. These calls operate like a read or write sequential of one character. They do not affect the column counter, nor do they provide special character handling (e.g., of carriage returns). These commands reference $TTI/$TTO or $TTI1/$TTO1; the console is always available to them, and you need no channel number or open command.

Get a Character (.GCHAR)

This command places a character typed on the console in AC0. RDOS right-adjusts the character (without parity) in AC0, and clears the left byte of AC0. You need no I/O channel for .GCHAR. RDOS will not echo the character on the console.

Format

```
.SYSTM
.GCHAR
error return
normal return
```

If no character is currently in the console input buffer, the system waits.

Possible error

AC2	Mnemonic	Meaning
3	ERICD	Console not in system.

Put a Character (.PCHAR)

This command types the character in bits 9-15 of AC0 on the console.

Format

```
.SYSTM
.PCHAR
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
3	ERICD	Console not in system.

Get the Input Console Name (.GCIN)

This command returns the name of the current console input device. This name is $TTI for the background program, and $TTI1 for the foreground program.

.GCIN and .GCOUT are useful in dual-ground systems because they allow each program to select the appropriate console for its ground at run time.

Required input

AC0 - Byte pointer to a six-byte area which will receive the console name.

Format

```
.SYSTM
.GCIN
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read into system area (unmapped only).
74	ERMPR	Address outside address space.

Get the Output Console Name (.GCOUT)

This command returns the name of the current output console: $TTO for the background program, and $TTO1 for the foreground program.

Required input

AC0 - Byte pointer to the six-byte area which will receive the console name.

Format

```
.SYSTM
.GCOUT
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read into system area (unmapped only).
74	ERMPR	Address outside address space.

Memory Allocation Commands

Excluding the Task Scheduler, and locations 0-15$_8$, RDOS resides in upper memory. It executes your programs in lower memory. Unmapped RDOS memory looks essentially as shown in Figure 3-5.

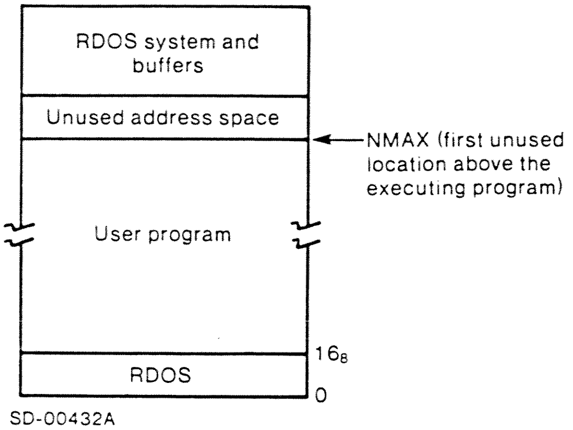


Figure 3-5. Unmapped Background Memory

The highest memory address available (HMA) is usually the first word below RDOS in an unmapped system. If, during loading, RLDR has placed its symbol table at the high end of user memory, HMA will be the first word below the table. The table will be in upper memory only if you include the global switch /S in the RLDR command. By default, RLDR loads it just above your program.)

Determine Available Memory (.MEM)

This command returns the current value of NMAX in AC1, and the value of HMA in AC0.

In unmapped systems, HMA represents the location immediately below the bottom of RDOS (or the bottom of the symbol table, if the program was loaded or bound with global /S). In mapped systems, HMA is the highest logical address available in the current program space.

Follow .MEM with a SUB 1,0 and INC 0,0 instruction to determine the amount of additional memory available to your program.

Format

```
.SYSTM
.MEM
error return
normal return
```

Possible errors

none

Change NMAX (.MEMI)

This commamd allows your program to increase or decrease the value of NMAX. The command updates the value of NMAX in the UST (in USTNM) and returns the new NMAX in AC1.

RDOS will not change NMAX if its new value would be greater than HMA+1. The system does not check NMAX against its original value (as determined by RLDR).

Whenever one of your programs will require memory space above its NMAX, it can invoke .MEMI to allocate the number of words needed. RDOS uses the value of NMAX to determine the amount of memory to save if it suspends a program. Generally, you should update NMAX even for temporary storage above the current NMAX. If you store a program without updating NMAX, the program may be suspended without enough information to continue. This is explained further in the discussion of program swaps, Chapter 4.

However, each of your programs should request only the memory space it actually needs, and should release memory space when it no longer needs it.

Required input

AC0 - The increment (positive) or decrement (in two's complement) of NMAX.

Format

```
.SYSTM
.MEMI
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
26	ERMEM	Attempt to allocate more memory than available.
74	ERMPR	Address outside address space.

Device Access Commands

This section describes the device access commands, .DEBL and .DDIS. In mapped RDOS systems, the map will trap if any user program attempts to access a system device (like the CPU or the floating-point unit). The call .DEBL gives a user program access to a system device, but you should use it carefully because it circumvents the map's safeguards. (Such instructions as INTDS or IORST will disable RDOS if access to the CPU is enabled.)

You must use .DEBL in any system if you have a hardware floating-point unit and are running foreground and background programs which both use floating-point arithmetic. In any such system, each program must enable access to the FPU so that the system can save and restore the FPU.

In a mapped NOVA system, the grounds can issue a .DEBL to device code 75 or 76. Either call will enable access to all three FPU devices (codes 74, 75, 76). Do not issue a .DEBL to device code 74 in a NOVA system.

In an ECLIPSE system, the programs can enable access to the FPU by issuing a .DEBL to device code 74 -- unless some other device (e.g., the I/O bus) is already wired as device 74, 75, or 76. If you have an ECLIPSE system in which both programs will not use floating-point arithmetic, and a device is wired as 74, 75, or 76, programs can access these devices if they use .IDEF (Chapter 7) instead of .DEBL.

Similarly, if your system has an optional integer MPY/DVD, and both programs want to use the MPY/DVD, they must enable access via .DEBL, then save and restore the MPY/DVD.

If you're in an unmapped system, in which two grounds won't access the FPU, the device access calls .DEBL and .DDIS are no-ops, and take the normal return.

If any system, we recommend that you .DEBL the FPU before using it.

Enable User Access of a Device (.DEBL)

This call permits your program to reference any device on a machine level, bypassing the normal system safeguards. Use it carefully. .DEBL is a no-op in unmapped systems (except as noted above, for hardware FPUs).

Required input

AC0 - Device code of the device you wish to access.

Format

```
.SYSTM
.DEBL
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
36	ERDNM	Device code exceeds 77_8.

Disable User Access of a Device (.DDIS)

This call is the complement of .DEBL. .DDIS prevents further machine-level access of a device in the system, thus restoring the system safeguards removed by a previous call to .DEBL. This call is a no-op in unmapped systems, except as noted above.

Required input

AC0 - Device code of the device to which you wish to disable user access.

Format

```
.SYSTM
.DDIS
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
36	ERDNM	Device code exceeds 77_8.

Read the Front Panel Switches or Register (.RDSW)

This system call allows your program to read the position of the front panel switches or the contents of the switch register. RDOS returns the switch configuration in AC0. Bit 0 equals switch 0, or the first number, etc.

Format

```
.SYSTM
.RDSW
error return
normal return
```

Possible errors

none

Clock/Calendar Commands

RDOS provides four commands to keep track of the time of day and the current date. It stores dates as days from December 31, 1967 (day 1 is January 1, 1968). RDOS uses a 24-hour clock. To set this clock, you must pass hours, minutes, and seconds in binary, in three accumulators.

Get the Time of Day (.GTOD)

This command requests the system to pass you the current time in hours, minutes, and seconds. RDOS will return the time in binary as follows:

AC0 - Seconds

AC1 - Minutes

AC2 - Hours (24-hour clock)

Format

```
.SYSTM
.GTOD
error return
normal return
```

Possible errors

none

Set the Time of Day (.STOD)

This command sets the system clock to a specific hour, minute, and second. You pass the initial binary values as follows:

AC0 - Seconds

AC1 - Minutes

AC2 - Hours (24-hour clock)

Format

```
.SYSTM
.STOD
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
41	ERTIM	Illegal time of day.

Get Today's Date (.GDAY)

This command requests the system to return the number of the current month, day and year. RDOS returns the month in AC1, the day in AC0, and the current year (less 1968) in AC2.

Format

```
.SYSTM
.GDAY
error return
normal return
```

Possible errors

none

Set Today's Date (.SDAY)

This command sets the system calendar to a specific date. The system will increment the date when the time of day passes 23 hours, 59 minutes, and 59 seconds. This routine works only on years from 1968 to 2099.

Required input

AC0 - Number of the day within the month.

AC1 - Number of the month (January is month 1).

AC2 - Number of the current year, less 1968.

Format

```
.SYSTM
.SDAY
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
41	ERTIM	Illegal day, month, or year.

Spooling Commands

SPOOL is an acronym for simultaneous peripheral operation on-line. RDOS automatically spools data output to the following devices: $DP0, $LPT, $LPT1, $PTP, $PTP1, $TTO, $TTO1, $TTP, $TTP1. You must explicitly enable spooling for a plotter ($PLT or $PLT1). During a spool, RDOS queues data on disk for one or more spoolable devices, making the CPU available for further processing while those devices receive the queued data. Spooling occurs only when no other system operations are ready; you control spooling by means of system commands .SPKL, .SPEA, and .SPDA.

Spooling requires that you SYSGEN at least 2 stacks in a single program environment, and at least 3 system stacks in a dual program environment (you allocate stacks at SYSGEN). RDOS will not spool with fewer stacks than these and all spooling commands will become inoperative. Spooling also requires disk buffering, and RDOS will allocate space for this dynamically from the master directory. If RDOS needs more disk space for spooling buffers and none is available, it will disable spooling (.SPDA). You can re-enable spooling later, when more disk space is available.

Stop a Spool Operation (.SPKL)

Use this command to halt a current spool operation. After you kill spooling on a device, you will lost all data on the output queue.

Required input

AC0 - Byte pointer to the name of the device receiving the spooled data.

```
.SYSTM
.SPKL
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
3	ERICD	Illegal command for device.
36	ERDNM	Device not in system.
74	ERMPR	Address outside address space.

Disable Device Spooling (.SPDA)

The .SPDA command stops a spoolable device from spooling its output. If you issue .SPDA while a device is spooling, RDOS will delay execution of the command until it has output all data waiting to be spooled. Data output to the device before the spooled data has been exhausted will itself be spooled to the output device, delaying execution of .SPDA even longer.

Required input

AC0 - Byte pointer to the device for which you are disabling spooling.

Format

```
.SYSTM
.SPDA
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
3	ERICD	Illegal command for device.
36	ERDNM	Device not in system.
74	ERMPR	Address outside address space.

Enable Device Spooling (.SPEA)

Use .SPEA to enable spooling on a device for which spooling has previously been disabled. RDOS itself may have disabled spooling because it lacked disk space; or you may have disabled spooling via .SPDA or CLI command SPDIS.

Required input

AC0 - Byte pointer to the device name.

Format

```
.SYSTM
.SPEA
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
3	ERICD	Illegal command for device.
36	ERDNM	Device not in system.
74	ERMPR	Address outside address space.

Keyboard Interrupts

You can interrupt the background program from the console by typing either CTRL and A or CTRL and C. To halt the foreground program from the background console, type CTRL and F. CTRL A and CTRL F work abruptly; they halt program execution in their respective grounds, save nothing, and give control to the higher-level program - generally the CLI. CTRL C writes the current core image to disk file BREAK.SV (FBREAK.SV if you issued the CTRL C from the foreground console), and gives control to the CLI. After RDOS has executed CTRL A, the message -INT will appear on the console; after CTRL C, the message BREAK will appear. For more on CTRL F, see Chapter 6.

If you want to program an interrupt, use the system call .BREAK; this produces the same effect as CTRL C. If, upon any of these interrupts, you want any program other than the CLI to gain control, you must set up its User Stack Table as described below.

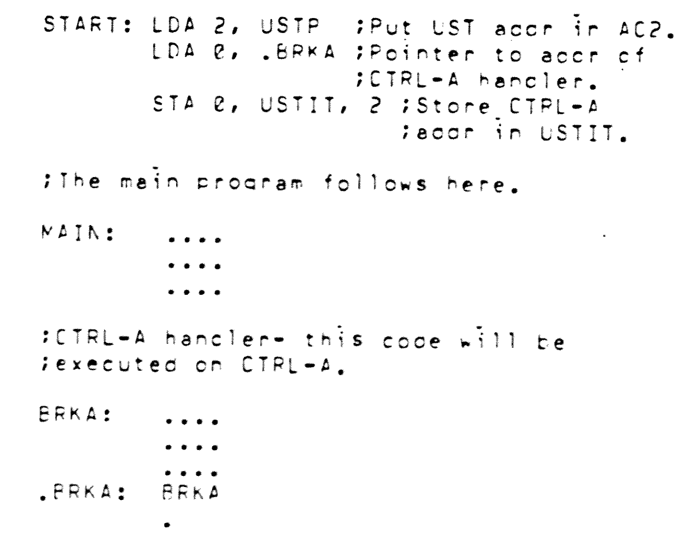
For each program level, the system creates a User Status Table (UST). Each UST is 24_8 words long, and resides in user address space, starting at location 400_8. Every UST includes two words, USTIT and USTBR, which contain addresses for CTRL A and CTRL C interrupt routines. USTIT contains the address of the routine which will gain control after you enter CTRL A; USTBR holds the address of the CTRL C routine. When you load a program, the loader initializes both words to -1, and you must change this if you want to specify your own routines. Chapter 5 describes the UST in detail.

If USTIT contains -1 when you hit CTRL A, or if USTBR holds -1 on a CTRL C or .BREAK, the system closes all channels on the current level and loads the next higher level program. This level's UST is then checked for the address of an interrupt routine. The system continues this process until it finds a program level whose UST contains the address of an interrupt routine. If it reaches the CLI on level 0, it uses the CLI's routine. But if you have CHAINed from the CLI, and the new level 0 program contains no interrupt routine address, the system will halt in Exceptional Status (see Appendix F).

During this search, the system checks each program level for a TCB queue. If the queue is missing (perhaps because you accidentally overwrote it; it is in user address space), the system skips this program and examines the next higher level program (see Figure 3-8).

After finding a program with USTIT or USTBR (as appropriate) containing an address instead of -1, RDOS checks another UST word, USTIA, to find a task's TCB address. The loader sets USTIA to 0 but it may contain a TCB address, as described below.

If USTIA contains 0, then the system appropriates the TCB of the current highest priority task (pointed to by USTAC), transfers that task's PC to temporary storage (TTMP in the TCB), and places the UST interrupt address in TPC (TPC is the program storage counter in the TCB). Control then goes to the scheduler, which starts the highest priority task. Since the UST interrupt address was placed in TPC of the highest priority task, RDOS executes the interrupt routine. (In a single task program, the program is the highest priority task.) Figure 3-6 shows a program with an interrupt handler.



```
START: LDA 2, USTP  ;Put UST accr ir AC2.
       LDA 0, .BRKA ;Pointer to accr cf
                    ;CTRL-A hancler.
       STA 0, USTIT, 2 ;Store CTPL-A
                       ;accr in USTIT.

;The main program follows here.

MAIN:    ....
         ....
         ....

;CTRL-A hancler- this cooe will be
;executed on CTRL-A.

BRKA:    ....
         ....
         ....
.BRKA:   BRKA
         .
```

Figure 3-6. Program with Interrupt Handler

If a task issued system call .INTAD before the interrupt, RDOS finds a nonzero value in USTIA (the issuing task's TCB address) RDOS then readies the issuing task, and stores the USTIT or USTBR address in the task's PC storage, TPC. It then disables further CTRL A or CTRL C interrupts, and passes control to the Task Scheduler. When the .INTAD task gains control, it executes the appropriate interrupt service, and reenables console interrupts -- if you desire -- by reissuing system call .INTAD or by issuing system call .OEBL, described below. Note that your main program should not issue .INTAD -- unless you want it to suspend itself. Figure 3-7 shows a program with an .INTAD task.

The BREAK file created by a CTRL C (or .BREAK) is a save file, containing the current state of main memory from SCSTR (the start of save files, location 16) through the highest of NMAX or the start of the symbol table, SST. RDOS places the break file in the current directory and uses the file name BREAK.SV (FBREAK.SV if the foreground program issued .BREAK; it deletes any existing (F)BREAK.SV first. If the system cannot write file (F)BREAK.SV (possibly because it lacks disk file space), control will go to one location before the address specified in USTBR, and AC2 will contain a system error code.

Although the (F)BREAK.SV is a snapshot of the current state of main memory, the file is not directly executable; it is generally useful for debugging. Before you try to execute it, you must consider how the CTRL C (or .BREAK) interrupt affected the system:

1) It closed all open channels, and you must reopen them if the breakfile requires them.

2) It purged all DELAY commands (yet their tasks remain suspended).

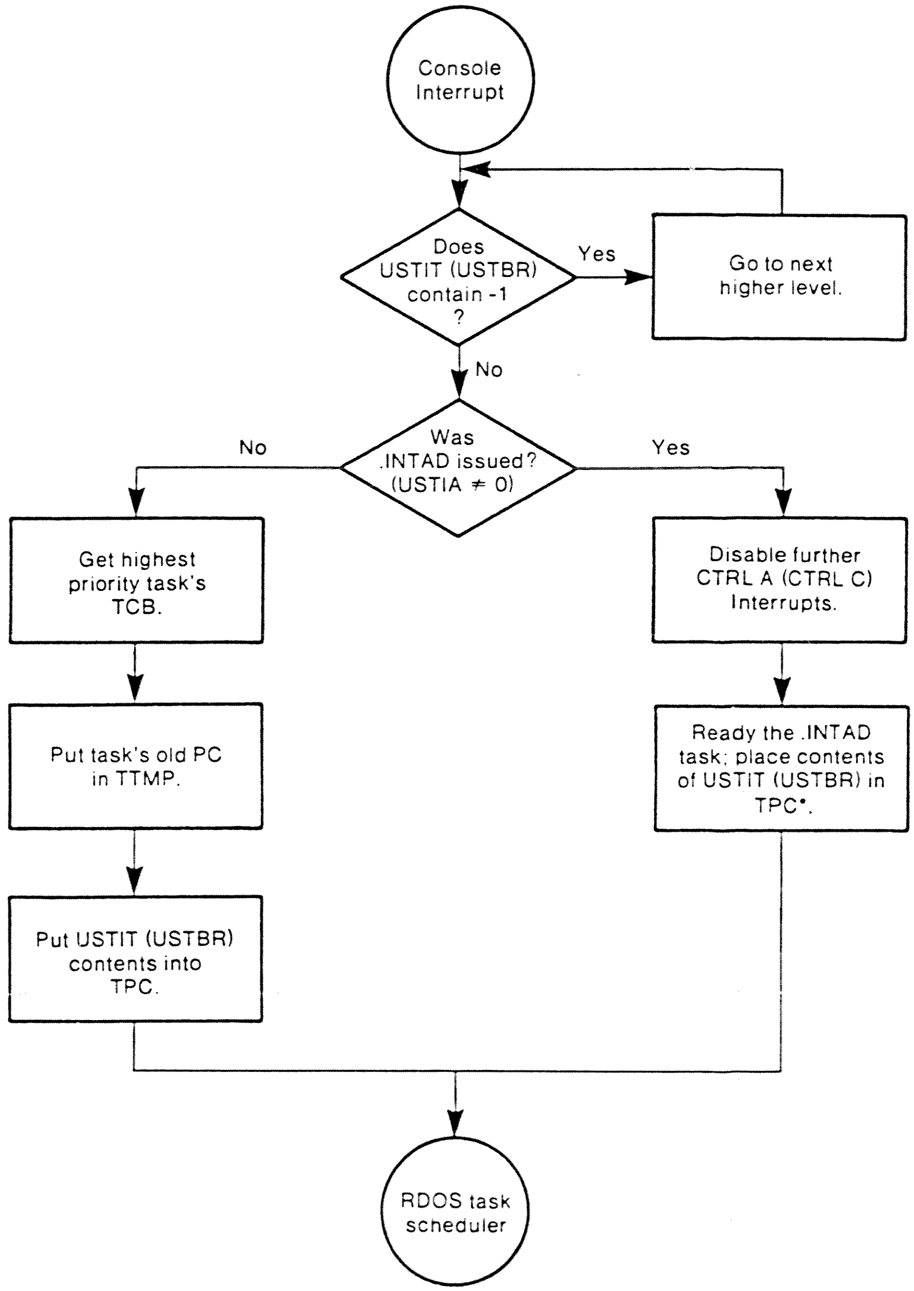
3) It removed all user-defined clocks and user interrupt devices; you must re-identify them if desired.

4) It destroyed all read-operator messages.

5) It disabled your access to all devices enabled via .DEBL, including the floating-point unit, and you must re-enable access if the breakfile will need these devices.

```
;The main task creates the INTAD task and
;initializes the ↑A processing address.

START:   SUB 0,0     ;0 priority for INTAD task.
         LDA 1, .INTSK  ;Addr of INTAD task.
         .TASK          ;Create the INTAD task.
         JMP ER         ;Mandatory.
         LDA 2, USTP    ;Put UST addr in AC2.
         LDA 0, .ROUT1  ;Name of ↑A routine.
         STA 0,USTIT,2  ;Put ↑A routine addr of
                        ;INTAD task in USTIT.
;The main program follows here.
MAIN:    ......
         ....

.INTSK: INTSK
         ....
.ROUT1: ROUT1

;Here is the INTAD task.
INTSK:   .SYSTM      ;On program execution, the INTAD
         .INTAD      ;task issues .INTAD, suspending
                     ;itself until ↑A is entered.
         JMP ER      ;System never takes error or
         JMP INTSK   ;normal return from .INTAD.
ROUT1:   ....        ;On ↑A, the INTAD task awakens
         ....        ;and executes this code.
         ....
         JMP INTSK    ;After doing its thing, the INTAD
                      ;task re-issues .INTAD, putting
                      ;itelf to sleep and re-enabling
                      ;↑A interrupts.

ER:      .SYSTM       ;(If program reads from console,
         .ERTN        ;error handler must pass EREOFs,
         JMP .        ;since ↑A,↑C supply EOFs to console.)

         .END START
```

Figure 3-7. Program with .INTAD Task


Console Interrupt
Does USTIT (USTBR) contain -1 ?
Yes
Go to next higher level.
No
Was .INTAD issued? (USTIA ≠ 0)
No
Yes
Get highest priority task's TCB.
Put task's old PC in TTMP.
Put USTIT (USTBR) contents into TPC.
Disable further CTRL A (CTRL C) Interrupts.
Ready the .INTAD task; place contents of USTIT (USTBR) in TPC*.
RDOS task scheduler

*If break fails, USTBR-1 is placed in TPC.

SD-00753

Figure 3-8. Program Interruption Logic Sequence

By default, when you execute a program, keyboard interrupts are enabled. RDOS provides two system calls to disable or re-enable further keyboard interrupts: .ODIS and .OEBL. These two calls do not affect the system call .BREAK (below), which performs the same operation sequence as CTRL C. To restore keyboard interrupts after .INTAD and *any* interrupt (CTRL A, CTRL C, or .BREAK), the .INTAD task must issue either .OEBL (below) or another call to .INTAD.

Interrupt Program and Save Main Memory (.BREAK)

System call .BREAK is operationally equivalent to typing CTRL C on the console; it saves the state of memory in save file format from location 16 to the highest of NMAX or the start of the symbol table, SST. The file name used is BREAK.SV (FBREAK.SV if .BREAK was issued by the foreground program). Any previous version of (F)BREAK.SV is deleted, and the breakfile is written to the current directory, where you can retain it, SAVE (CLI command) it under another name, or delete it. Generally, because system breaks close all channels, the breakfile is useful only for debugging with a disk editor, such as OEDIT or SEDIT.

The memory image file created by CTRL-C and .BREAK saves the program in the following state:

1) all open channels are closed:

2) all .DELAY commands have been purged (yet their tasks remain suspended);

3) user-defined blocks, user interrupt devices and user device enables (.DEBL) are removed;

4) all read-operator messages are lost.

5) all user accesses enabled via .DEBL are lost.

Unlike the CTRL C interrupt mechanism, the .BREAK call is operative at all times and is not disabled by the .ODIS command.

If USTBR (see preceding section) contains a valid address, control goes to this address after RDOS writes (F)BREAK.SV to disk. If USTBR contains -1, control will return to the next higher level program and RDOS will examine its USTBR. Control eventually goes to the first higher level program whose USTBR contains a valid address. If RDOS cannot write the break save file (e.g., due to insufficient file space), control goes to one location before the address contained in USTBR.

Format

```
.SYSTM
.BREAK        ;No standard error
              ;or normal returns
```

Possible errors

AC2	Mnemonic	Meaning
27	ERSPC	Out of disk space.
60	ERFIN	BREAK.SV (or FBREAK.SV) is in use.
101	ERDTO	Disk timeout occurred.

Disable Console Interrupts (.ODIS)

Use this command to disable console interrupts within your program. When you issue .ODIS from the background, it disables CTRL A and CTRL C interrupts. When you issue .ODIS from the foreground, it disables CTRL A and CTRL C interrupts and the background CTRL F interrupt. However, you can never disable the .BREAK system command with this command. You can re-enable console interrupts by issuing system call .OEBL from your program.

Format

```
.SYSTM
.ODIS
error return
normal return
```

Possible errors

none

Enable Console Interrupts (.OEBL)

When you first bootstrap a system, RDOS enables console interrupts CTRL A, CTRL C amnd CTRL F. If you disable console interrupts by system call .ODIS or by processing a console interrupt and system call .ODIS or by processing a console interrupt with an .INTAD task, this call re-enables them within its program environment.

Format

```
.SYSTM
.OEBL
error return
normal return
```

Possible errors

none

Reserve a Program Interrupt Task (.INTAD)

This system call enables keyboard interrupts and permits you to assign a task to service CTRL A, CTRL C and .BREAK program interrupts. The task that will service these interrupts must issue .INTAD; RDOS will recognize it as the interrupt ask. The main task should not issue .INTAD. RDOS uses the .INTAD task (instead of a program task's TCB) for interrupts, hence it perserves the current program environment (aside from any system call executing when the interrupt occurs). For more on .INTAD, see the preceding section.

Format

```
.SYSTM
.INTAD
error return
normal return
```

Possible errors

none

End of Chapter

Chapter 4
Extended User Address Space: Swaps, Chains, User Overlays, Window Mapping, and Extended Memory I/O

Occasionally, one of your programs will require more memory than is available in the machine. This chapter explains how to extend the limits of main memory with disk space. It has two main sections:

- Swaps, chains, and User Overlays - This material applies to all systems and applications; you must understand these concepts to write advanced programs in RDOS.

- Memory protection, Virtual User Overlays, Window Mapping, and Extended Direct-Block I/O - you may want to exploit these programming tools if you have a mapped system.

Program swaps, chains and user overlays effectively extend main memory with disk space. When a program swaps or chains, it calls another program into execution. During this process, the same areas of your address space can do many different things.

Programs can execute a swap from one of four RDOS levels of control, where one level calls another. *Chained* programs are called in sequence by a program on the same level, and overwrite the calling program. *Overlays* also operate on one level, but they are called in succession by a root program in core and placed in a reserved area (node) in core.

Swaps and chains are described below; you will find user overlays in the next section.

When you plan program *swaps* you should ensure that the NMAX memory in use accurately reflects the core for every program in use; if it does not, part of the calling program might be lost. Upon a program swap, RDOS saves the current core image up to the higher of NMAX or SST (start of the user symbol table). It is very important that your program does not use temporary storage above its original value of NMAX at load time without instructing the system first to allocate more memory (see .MEMI, Chapter 3) for this space. If your program exceeds NMAX and invokes another program, RDOS will not save part of the calling program's memory state. Even if the executing program does not call another program, a BREAK from your console may force suspension. To avoid these problems, NMAX must always correctly reflect the core in use.

The operations of swapping, chaining, and returning halt activity in the current program. RDOS terminates calls and conditions that would not be appropriate in the new program (most of these involve multitask activity). The system terminates the following calls and conditions when a change of program occurs. Many of the calls are detailed elsewhere in this manual, and you should read the indicated sections if you want more information.

1) A *return* or *chain* closes all channels, and the new program must open the channels it requires; as described under I/O commands, Chapter 3.

2) All STTI (STTI1) input is halted; this applies to such calls as .GCHAR (Chapter 3), .TRDOP (Chapter 5), and .RDOP (Chapter 6).

3) Any system devices enabled for user access via .DEBL (Chapter 3) are disabled; thus the new program must enable access to the hardware floating-point unit (if you have one).

4) Console interrupts are enabled, removing any outstanding disable calls by .ODIS (Chapter 3).

5) The state of the floating point unit is not preserved.

6) All interrupt mesage transmissions, .IXMT (Chapter 5), are removed.

7) If you have defined a user clock (.DUCLK, Chapter 5) or a system delay (.DELAY, Chapter 5), it is removed.

8) If your system has operator messages, the state of the OPCOM (Chapter 5) is lost.

9) All user-defined interrupt service (.IDEF, Chapter 7) is removed, as is any mapped system data channel map setting (.STMAP, Chapter 7).

10) Mapped only. All memory write-protection definitions, .WRPR, are removed.

11) Mapped only. Extended space reserved for virtual overlays is released.

12) Mapped only. All extended memory definitions made via window mapping are removed.

13) Mapped only. Any dual-program communications area defined via system call .ICMN (Chapter 6) is removed.

Program Swapping and Chaining

This section describes the swapping and chaining call .EXEC, .RTN and two swap return calls: .RTN and .ERTN. It also describes the overlay replacement call, .OVRP.

Any program executing under RDOS can suspend its own execution and swap in another program, or chain to an executable segment of itself. Programs with open multiplexor lines must close them before swapping ~~or~~ they will take the error return from .EXEC.

Program swaps can exist in up to five levels, where one level calls for another and the Command Line Interpreter exists at the highest level, level 0. The CLI is merely one program which RDOS can execute. Its only special property is that it normally executes at the highest level in the system. Normally, the system utility programs supported by the CLI (e.g., the Text Editors, the assemblers, and the binder or loader) execute at level 1. When you execute a program or system utility from the CLI, RDOS often swaps the CLI to disk and calls it back automatically after execution via .RTN.

You can also write a large program in a sequence of executable segments, where the end of each segment evokes the beginning of the next segment, and the series ends by evoking the CLI. This process is called *chaining*, and it all occurs on one level. The length of the entire program is limited only by the available disk space. You can begin the execution of a program chain by either .EXEC, or, from the console, via the CLI CHAIN command.

When your program issues the .EXEC call, a program *swap* or *chain* occurs (you specify which in AC1). For a swap, RDOS saves a core image of the current program and brings the new program specified in AC0 into main memory and executes it. The calling program's task control block (TCB) saves its accumulators, Carry, and PC. The new program can swap itself and execute the old program by issuing the call .RTN (or .ERTN); or it can swap to a lower level by issuing a .EXEC call. Any program can check its current level by issuing system call .FGND (Chapter 4).

Occasionally, we will use the term "push" instead of "swap". Each term means the same thing: execute a program on the next lower level via .EXEC. The CLI command POP, which instructs RDOS to execute the program on the next higher level, corresponds roughly to system call .RTN.

If AC1 specifies a *chain*, RDOS does not save the core image, and brings the program specified in AC0 into core and executes it. The new program can .EXEC the old program (or any other) into execution when it is finished.

Any program you plan to swap or chain must be an executable save file.

When the calling program's execution resumes after a *swap* all channels which were open when the swap occurred will be open. To restore the other conditions (2 through 12 above) to the caller, you must use the appropriate system or task call.

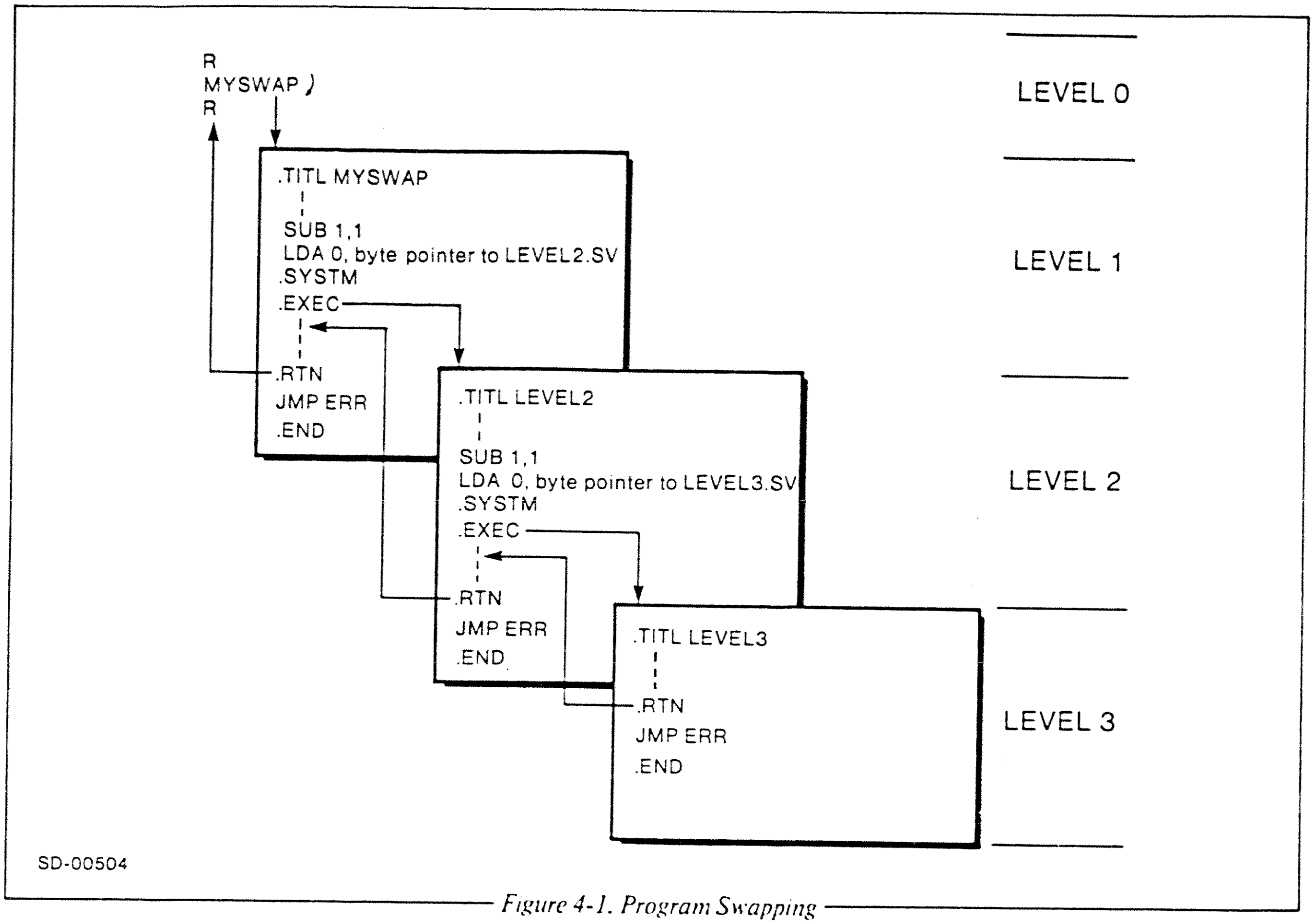


Figure 4-1. Program Swapping

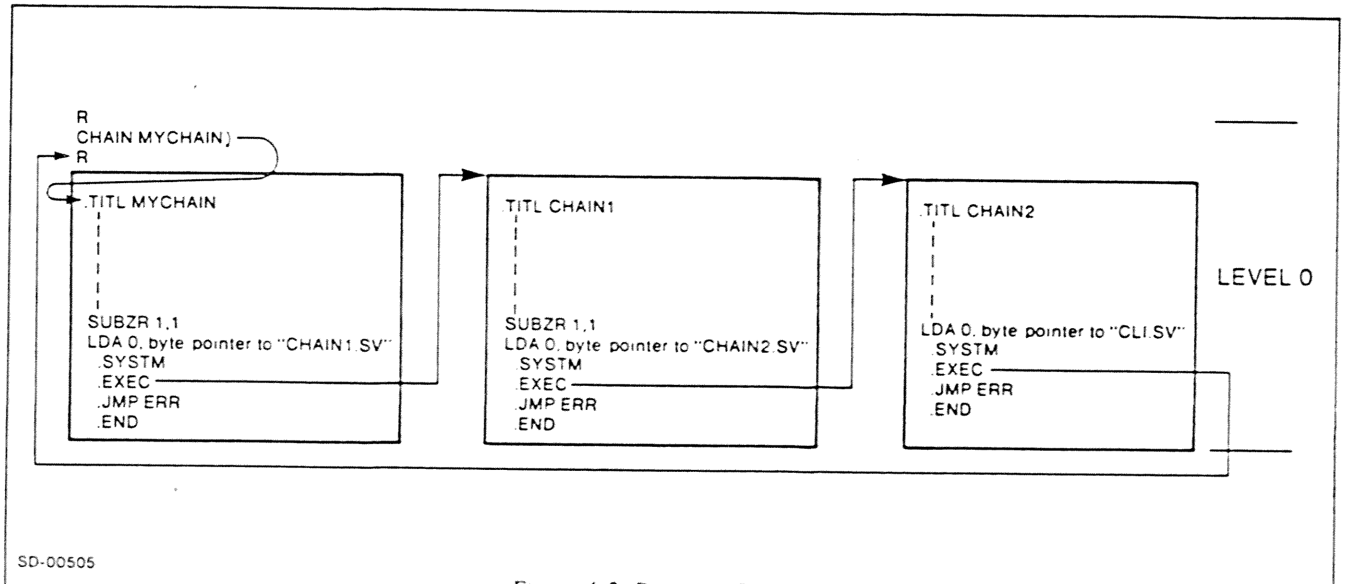


Figure 4-2. Program Chaining

Swap or Chain a Save File into Execution (.EXEC)

This command requests the system to swap or chain a program. See Figures 4-1 and 4-2 for illustrations of each.

Required input

AC0 - Byte pointer to save filename of new program.

AC1 - Specifies a code for swap or chain (see below for code).

The code in AC1 indicates one of two starting addresses: the program starting address (USTSA), and the Debug III starting address (USTDA). See Chapter 5, User Status Table, for descriptions of the addresses.

If bit 0 of AC1 is 1, RDOS will not save the current level, and the operating level will remain unchanged. This feature provides unlimited program chaining.

Note that you cannot swap from the foreground of an unmapped system. If an unmapped foreground program tries to swap, RDOS will return error 25 (ERCM3). You *can*, however, chain from an unmapped foreground, if the new program has the same or a smaller memory requirement than the old.

The permissible codes input in AC1 are:

Code	Meaning
0	Swap to user program. Control goes to the highest priority ready task.
1B0	Chain to user program.
1	Swap and start at debugger address.
1B0+1B15	Chain and start at debugger address.

Note that the new program will receive the contents of AC2. If the new program is the CLI (CLI.SV), and AC2 contains a nonzero value, the CLI will search its special command file CLI.CM for commands. This mechanism is fully described in an appendix of the CLI manual.

Format

```
.SYSTM
.EXEC
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
4	ERSV1	File requires save attribute (S).
12	ERDLE	File does not exist.
25	ERCM3	More than 5 swap levels or swapping from unmapped foreground.
26	ERMEM	Attempt to allocate more memory than is available.
32	ERADR	Illegal starting address.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
73	ERUSZ	Too few channels defined at load time or SYSGEN time.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
125	ERNSE	Program not swappable.

Note: RDOS will return ERADR (32) status if:

1) No starting address was specified for the save file and code 0 is given (i.e., bit 15 is reset to 0).

2) The Debugger was not loaded as part of the save file and code 1 is given (i.e., bit 15 is set to 1).

Return to the Next Higher-Level Program (.RTN)

.RTN closes all open channels and returns to the calling program at its normal return point. All the calling program's accumulators are restored, and control passes to the instruction at the return point. If the level 0 foreground program issues .RTN, RDOS will close all foreground channels and release the foreground. The message FG TERM will appear on the background console.

Format

```
.SYSTM
.RTN
error return
```

The normal return is impossible, since RDOS restores the calling program in memory. The error return is reserved for compatibility with RTOS but is never taken. Error conditions cause Exceptional System Status (see Appendix F).

Return from Program Swap with Old Program's Error Status (.ERTN)

This command instructs a called program to return error information to the calling program. Use it for error returns when you want to know why a swapped program took the error return.

This call is identical to .RTN except that normal return is made to the error return of the higher level program; upon return, AC2 contains the lower-level program's AC2 instead of the higher-level program's AC2. A single word of status can, therefore, be returned. If a program issuing a .ERTN has been executing at level 1 (and is returning to the CLI) the CLI will output an appropriate message concerning the status code in AC2. (If the CLI recognizes the code as a system error code, it will print a text message. ERDLE (12) would evoke the message FILE DOES NOT EXIST.) If RDOS returns *null error* ERNUL (20) in AC2, the CLI will report no error message.

Note that if the called program passes EREXQ (17) in AC2, the CLI will take its next command from disk file CLI.CM. This mechanism is described in an appendix of the CLI manual. If the CLI does not recognize the code, RDOS will type out the message UNKNOWN ERROR CODE *n*, where *n* is the numeric code in octal.

Format

```
.SYSTM
.ERTN
error return
```

The error return is reserved for compatibility with RTOS but is never taken. Error conditions cause Exceptional System Status (see Appendix F).

User Overlays

User overlays are blocks of code, placed in an overlay file, that support a root program. This root program is a save file that remains in memory throughout a program level; it extends from location 16_8 to NMAX, and calls overlays from disk into core as required. The overlay file is a contiguous file, and is divided into segments. Each segment contains the overlays which the root program will load one at a time into a reserved area of memory called a *node*. The RLDR command loads the root program, creates the overlay file, places overlays into segments of the file, and establishes the core node size. If you specify overlays in the RLDR command line, RLDR produces a save file, named *filename.SV*, and an overlay file named *filename.OL*. Unless you specify otherwise with switches, *filename* is the name of the first binary in the command line.

To use overlays, your program must open the overlay file on an DOS channel (.OVOPN n); it must then instruct DOS to load (.OVLOD n) one overlay at a time from a segment into its node. The node is reserved for the overlays in its segment until the program terminates. Your program can free the channel by .CLOSEing it. (This process differs slightly for a multitask program; if you plan a multitask program, see "User Overlay Management" in Chapter 5.) The EXAMPLE program in Appendix D shows a root program supporting 2 overlays.

The size of each node is the smallest multiple of 400_8 words large enough to contain the largest overlay in the node's segment. If any overlay is not exactly the size of its node, it will be padded out with zeroes. This means that any segment size equals the node size multiplied by the number of overlays within the segment. Each segment is identified on disk by its corresponding node number.

Each overlay file is a contiguous disk file and can hold up to 124 overlay segments. You can place no more than 125 overlays in a segment, and no overlay can be larger than 126 disk blocks (31,256 words). If the overlays in a segment differ significantly in size, a lot of disk space will be used to pad out the smaller overlays to the standard size; the same amount of memory will be used to pad out the core node. Therefore, if you can, you should place overlays of roughly the same size in the same segment.

Directory information for each overlay resides in an overlay directory, which RLDR builds into the program's save file (see Appendix E). Each overlay has a label which the system uses to identify it; this label resolves to a node number and an overlay number, packed by half-words.

The format required for creating an overlay file and associating it with a root program is shown in Chapter 4 of the CLI manual under the RLDR command. The following discussion extends the sample commands presented there:

```
RLDR R0 [A,B,C,D] R1 R2 [E,F G,H])
```

As illustrated in Figure 4-3, this command creates a disk save file, R0.SV, and an overlay file, R0.OL. The file contains R0, R1, and R2; it also contains vacant areas (nodes) for the overlays in each segment. The overlay file contains seven overlays: binary versions of A, B, C, D, E, F G, and H. These overlays are grouped in two segments according to the brackets. Segment 0 of overlay file R0.OL contains overlay A (number 0, for node 0), overlay B (number 1 for node 0), overlay C (number 2 for node 0), and overlay D (number 3 for node 0). Segment 1 of R0.OL contains overlay E (number 0 for node 0), overlay F and G (number 1 for node 1), and overlay H (number 2 for node 1). Note that the order in which you give the overlay binaries in the command line determines both the overlay number and node number of each overlay.

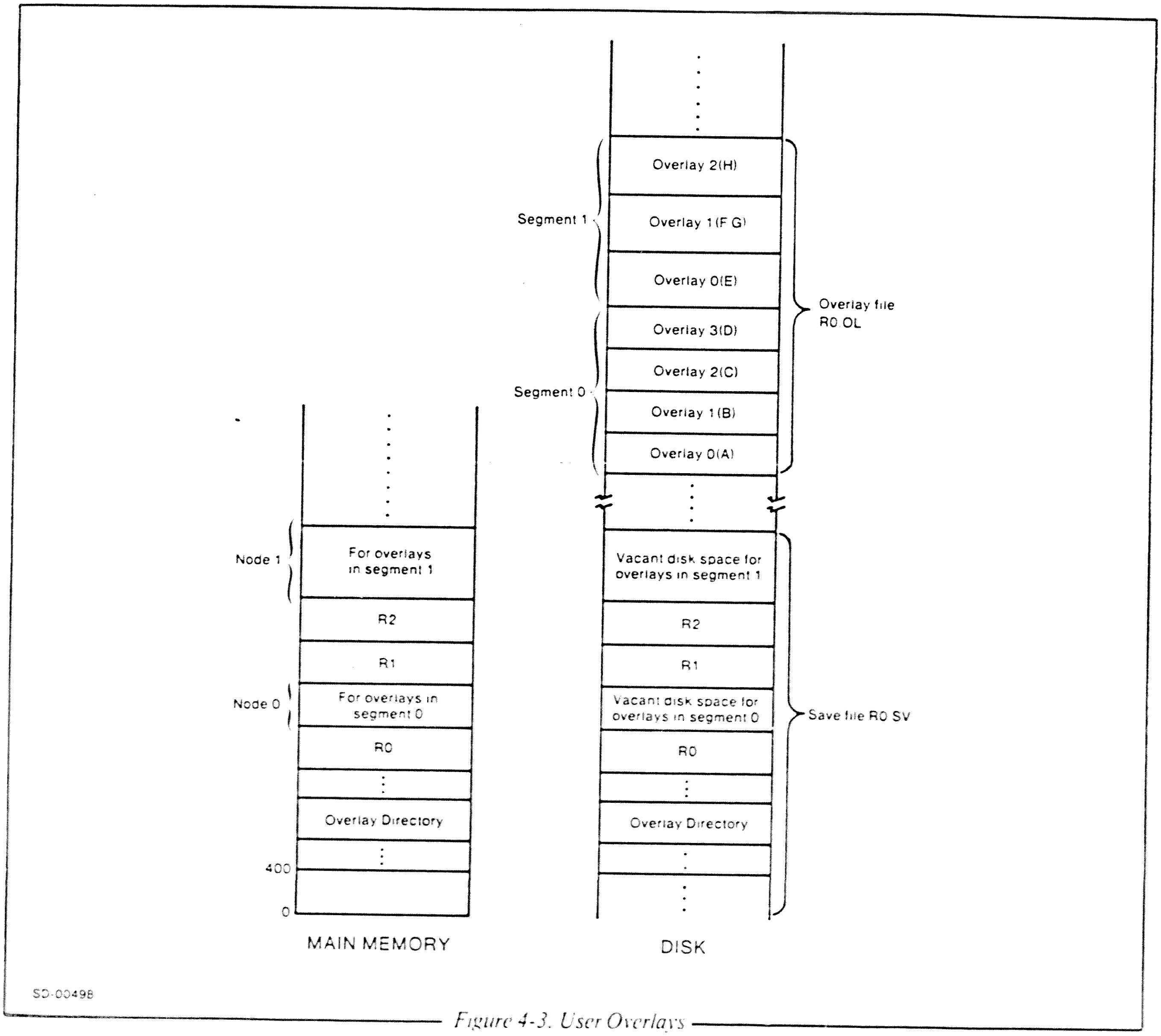


Figure 4-3. User Overlays

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number
1	ERFNM	Illegal file name
6	EREOF	End of virtual overlay (mapped files only).
7	ERRPR	Attempt to read open a read- protected overlay node (mapped only).
12	ERDLE	Nonexistent file.
21	ERUFT	Atempt to use a channel which is already in use.
26	ERMEM	Insufficient memory to load (.OVLD or .TOVLD) virtual overlays (mapped only).
30	ERFIL	File read error or virtual overlay file (mapped only-mag tape (bad tape)).
40	EROVA	Not a contiguous file (virtual overlays - mapped only).
53	ERDSN	Nonexistent file.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).

Load an Overlay (.OVLOD)

This command loads an overlay into its reserved memory node.

There are 2 types of overlay loads: conditional and unconditional. An unconditional load loads an overlay whether the overlay is in memory or not. This guarantees a fresh copy of the overlay (except for virtual overlays). A conditional overlay request, on the other hand, loads an overlay only if it is not already in memory. The conditional request can save you time, but you should use it for reentrant overlays only.

The .OVLOD command will load the overlay conditionally if you set AC1 to 0; or unconditionally if you set it to -1. We recommend that all your overlays be reentrant; if any overlay is not, be sure to load it unconditionally.

Required input

AC0 - Overlay node value in the left byte, and the overlay number value in the right byte. Or, if you used .ENTO, symbolic name, as explained under *User Overlays.*

AC1 - If 0, load conditionally; if -1 load unconditionally.

Format

```
.SYSTM
.OVLOD n          ;Load overlay opened on channel n
error return
normal return
```

In a multitask environment, only one task can issue .OVLOD commands. (See "User Overlay Management," .TOVLD command in Chapter 5 for more on multitasking.)

Under certain conditions (such as a nonmatching save and overlay file), the left byte of AC2 may be nonzero on an error return.

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	File not opened.
30	ERFIL	Read error (tape).
37	EROVN	Illegal overlay number.
40	EROVA	Overlay file is not a contiguous file.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

Replace Overlays in an Overlay File (.OVRP)

Although the RLDR utility can create an overlay file, it cannot modify that file. You can, however, create a replacement overlay file with the CLI OVLDR utility and execute the replacement with the call .OVRP (or CLI command REPLACE).

When creating the new overlay file with OVLDR, you make the changes you want, and give the new file the same name as the old file. The CLI appends the extension .OR to this name. The old file is not affected by the execution of the OVLDR command: it remains the current overlay file until you execute either .OVRP or REPLACE.

Even if both grounds are using the old overlay file, your program can update it via .OVRP without halting the programs that are using it. (See OVLDR in the CLI manual for more detail.)

Required input

AC0 - Byte pointer to overlay replacement file name (savefilename.OR)

AC1 - Byte pointer to overlay file name. (savefilename.OL)

Format

```
.SYSTM
.OVRP
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
6	EREOF	End of file.
12	ERDLE	One or both files do not exist.
27	ERSPC	Out of disk space.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout.

Protecting User Memory Under Mapped RDOS

If you have a mapped system, your programs can write-protect 1K blocks of memory with the .WRPR call. By default, RDOS does not protect memory; after you write-protect it, it remains protected until you issue the call .WREBL (or execute a new program).

Write protection prevents system read calls (which read from a file and write to a specified address - e.g., .RDL, .RDB) from writing into the protected block; it also prevents such instructions as STA from writing to these block(s). .WRPR does *not* prevent your program from loading overlays, or swapping or chaining a new program into the protected blocks.

In RDOS, a memory block is $1,024_{10}$ (1K) words, unlike a disk block, which is 256_{10} words. RDOS allots all mapped memory to programs (via CLI command SMEM) in 1,024-word blocks; hence .WRPR write-protects memory in 1,024-blocks. If an area you want to write-protect extends across a 1024 word boundary, RDOS write-protects both blocks.

For greater code integrity, you can write-protect your overlay nodes. You must do this carefully, however, to protect *only* the nodes; if your program inadvertantly write-protects other areas, it may not be able to run properly.

Luckily, RLDR reserves overlay nodes in integer multiples of 400_8 words, and you can use these multiples to help align your write-protection. The following examples show how you might do this (and what might happen if you didn't bother). Assume that you are about to bind/load a program which will have one overlay segment, and include 3 overlays. You contemplate (but don't type) this command:

```
RLDR R0 R1 R2 [A,B,C D]
```

You then proceed to check the sizes of all binaries with the Library File Editor (LFE). R0, R1, and R2 require 3600_8 words, which you round up to 4000_8; A and B are each 1000_8, and C D is 1500_8 words. The loader reserves an overlay node for the third (i.e., the largest) overlay, which is 1500 words. Therefore the node will be 2000_8 words long (1500_8 exceeds $3*400_8$).

This node size fits nicely in a memory block (2000_8 = $1,024_{10}$ = 1K); You will need to write-protect only one block, if you align it properly.

Misalignment of the overlay node will write-protect blocks outside the overlay and this would be undesirable.

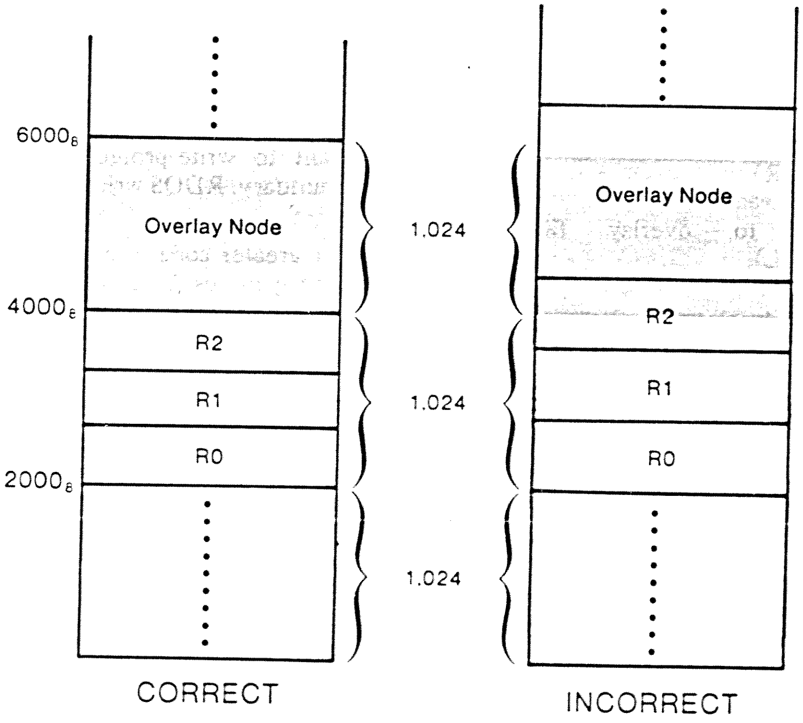
You can align the node for future write protection by judicious use of the RLDR local switch /N. The following command illustrates the use of local switch /N:

```
RLDR R0 2000/N R1 R2 4000/N [A,B,CD] )
```

/N forces the NREL pointer to the specified octal value. RLDR builds NREL upward from the bottom of user space for each binary loaded. The NREL figure pertains to the file whose name follows the switch. See RLDR in the CLI manual for more detail.

As shown in Figure 4-5, locations 4000_8 through 6000 would be reserved for the overlay node. There's enough room in R0 to insert a .WRPR instruction which will protect the node, without affecting the rest of the save file.

```
        LDA 0, LA     ;THE LOWER ADDRESS
        LDA 1, HA     ;THE HIGHER ADDRESS
        .SYSTM
        .WRPR
        JMP ER
.
.
.
LA:4000
HA:5777
```



Shading indicates memory protection.

Figure 4-5. Write-Protecting Memory

Protect a Memory Area from Modification (.WRPR)

By default, RDOS write-enables all memory blocks. This system call write-protects contiguous sections of mapped memory, as specified in AC0 and AC1. The blocks you specify will remain write-protected until your program removes this protection via .WREBL, until it issues an .EXEC, .RTN, or .ERTN, or until you enter a keyboard interrupt.

RDOS protects memory in 1024_{10} - word blocks, just as it allocates mapped memory in 1024-word blocks. If the addresses you specify cross a block boundary, RDOS will write-protect both blocks entirely.

.WRPR is a no-op in unmapped systems, and takes the normal return.

Required input

AC0 - Lower address of the series to become protected.

AC1 - Higher address of the series.

Format

```
.SYSTM
.WRPR
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
74	ERMPR	Illegal address.

Remove the Write-Protection from a Protected Memory Area (.WREBL)

This call removes the write-protect restriction from a memory block or a series of blocks. Memory which you write-protected by a call to .WRPR was protected in 1024_{10} word blocks: .WREBL also write-enables in 1024_{10} word blocks. If the addresses you specify cross a block boundary, RDOS will write-enable all addresses in each block.

Required input

AC0 - Lower address of the series to be write-enabled.

AC1 - Higher address of the series.

Format

```
.SYSTM
.WREBL
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
74	ERMPR	Illegal address.

Extending Your Address Space Under Mapped RDOS

In any Data General computer, mapped or unmapped, the directly-addressable memory available to a program cannot exceed 32K words. Naturally, this depends on the total amount available in the machine; in a dual-program environment, each of two programs can use up to 32K of this directly-addressable space. In a mapped system, directly-addressable space is called logically-addressable space.

Mapped RDOS, however, permits a program in either a single or dual environment to access memory outside its logical address space. Memory outside logical address space is called *extended address space* or *extended memory*.

You allot the total (both logical and extended) address space to a program via the CLI SMEM command.

Mapped RDOS offers two programming tools for using extended address space: *window mapping* and *virtual overlays*. Window mapping also allows you to transfer 256-word blocks of data via extended direct/block I/O.

Virtual overlays (like conventional overlays) are most useful for storing your subroutines; you define them via RLDR. Window mapping is most useful for extended data storage; your program defines a window map. You can use both features in one program.

Virtual Overlays

Virtual overlays provide one means of using extended address space. The major difference between ordinary user overlays and virtual overlays is that the former are disk-resident and permit only one memory resident overlay at a time from any segment while all virtual overlays are resident simulaneously in extended address space.

After you have built a virtual overlay file into your program, your program handles it as an ordinary overlay file. That is, both the normal and virtual overlay files are contained in the overlay (.OL) file which you must open (via .OVOPN) before you can access *any* overlay within the file. You must load each virtual overlay, like each normal overlay, via .OVLOD or .TOVLD (Chapter 5), before your program can use it. Multiple tasks can share a virtual overlay reentrantly. When all tasks have released the overlay (.OVREL), another task can use the overlay node. Loading a virtual overlay is quicker than loading a normal overlay, since the former requires only a memory remap operation, while the latter requires a disk access. Note that you cannot "refresh" virtual overlays by reloading them.

You can define a virtual overlay with the RLDR load /V switch:

```
RLDR root program ... [virtual overlay,...]/V
```

Virtual overlays must precede conventional overlays in the RLDR command line. Space for each virtual overlay is allocated in 1K ($1,024_{10}$) word pages (RLDR pads unused space). Each page begins on a 1K boundary (from page 0).

The virtual overlay node always occupies logical address space, and it always holds the first virtual overlay in the RLDR command line when you open the overlay file. Other virtual overlays occupy extended address space. When the program .OVLODs (.TOVLDs) another virtual overlay, the new overlay remaps into logical space, and the original overlay remaps into extended space. Thus the amount of extended space which RDOS uses for each virtual overlay segment will be:

```
(number-of-virtual-overlays-in-segment-1)*(node-size)
```

As mentioned above, .OVLODing (.TOVLDing) a virtual overlay causes a remap, while .OVLODing (.TOVLDing) a conventional overlay requires a disk access.

Virtual overlays release extended address space only when the program performs a program swap, chain or return (.EXEC, .RTN/.ERTN); therefore if your program has virtual overlays open, and it will swap, it should .CLOSE them before the swap and reopen them when it returns.

For example, the following command:

```
RLDR MAIN [VW,X,Y,Z]/V [A,B,C]
```

creates save file MAIN.SV, and overlay file MAIN.OL; MAIN.OL contains binaries A, B, and C as conventional overlays, and V W, X, Y, and Z as virtual overlays. When MAIN opens the overlay file, RDOS will use the map to set up a pointer from the virtual node to overlay V W. The overlay open command allocates all extended memory required for virtual overlays, and loads them from disk into this memory.

RDOS will do nothing with the conventional overlay node. Memory and disk will look like Figure 4-6.

Now, assume that MAIN opened overlays on channel 3, used virtual overlay V W, and wanted to use virtual overlay Z:

```
.
LDA 0,OVZ  ;OVZ HAD BEEN ASSIGNED
           ;VIA .ENTO.
SUB 1,1    ;VIRTUAL OVERLAYS ARE
.SYSTM     ;ALWAYS LOADED
           ;CONDITIONALLY.
.OVLOD 3   ;LOAD VIRTUAL OVERLAY Z.
.
           .
```

V W would remap into extended memory, and Z would remap into logical address space, as shown in Figure 4-7.

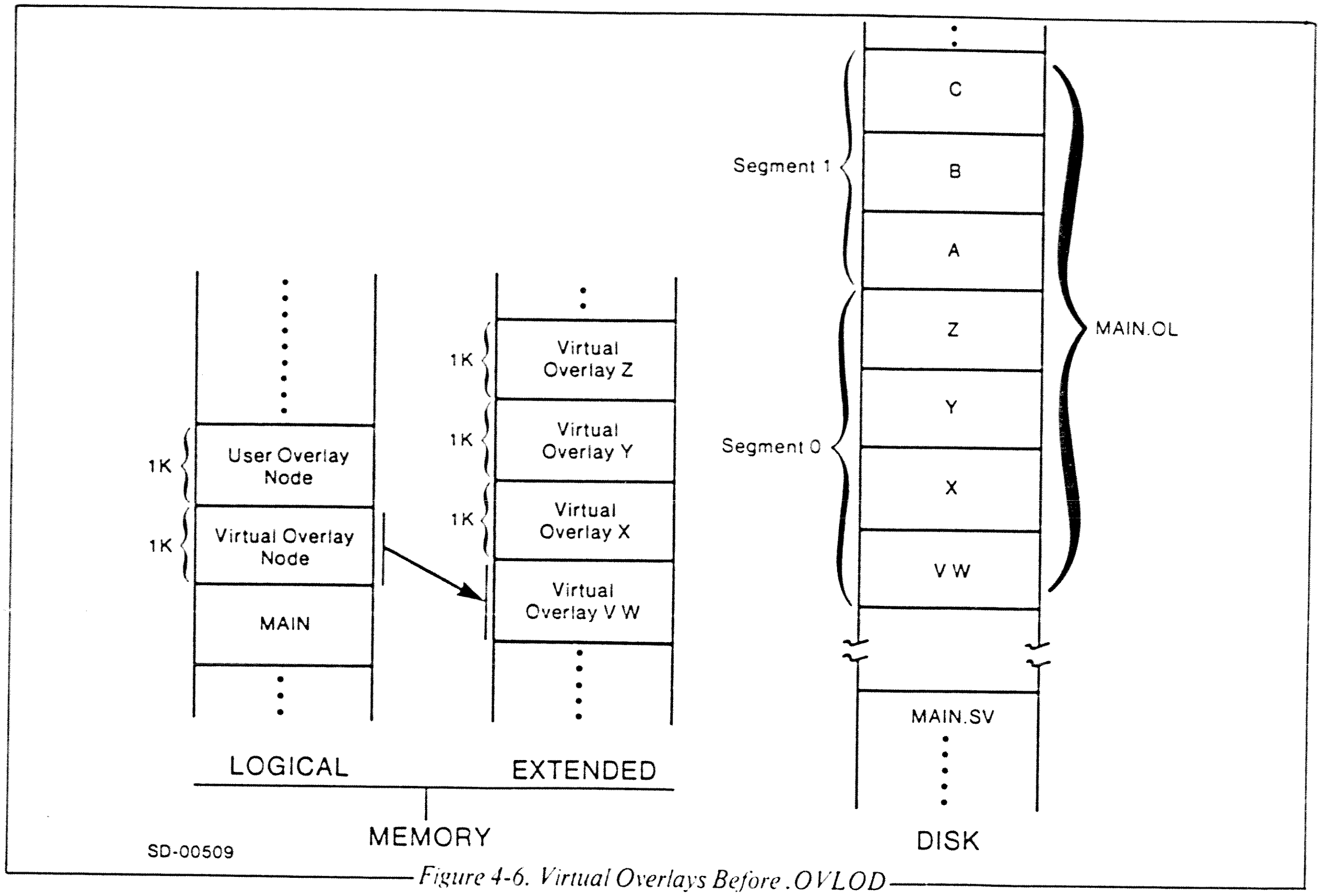


Figure 4-6. Virtual Overlays Before .OVLOD

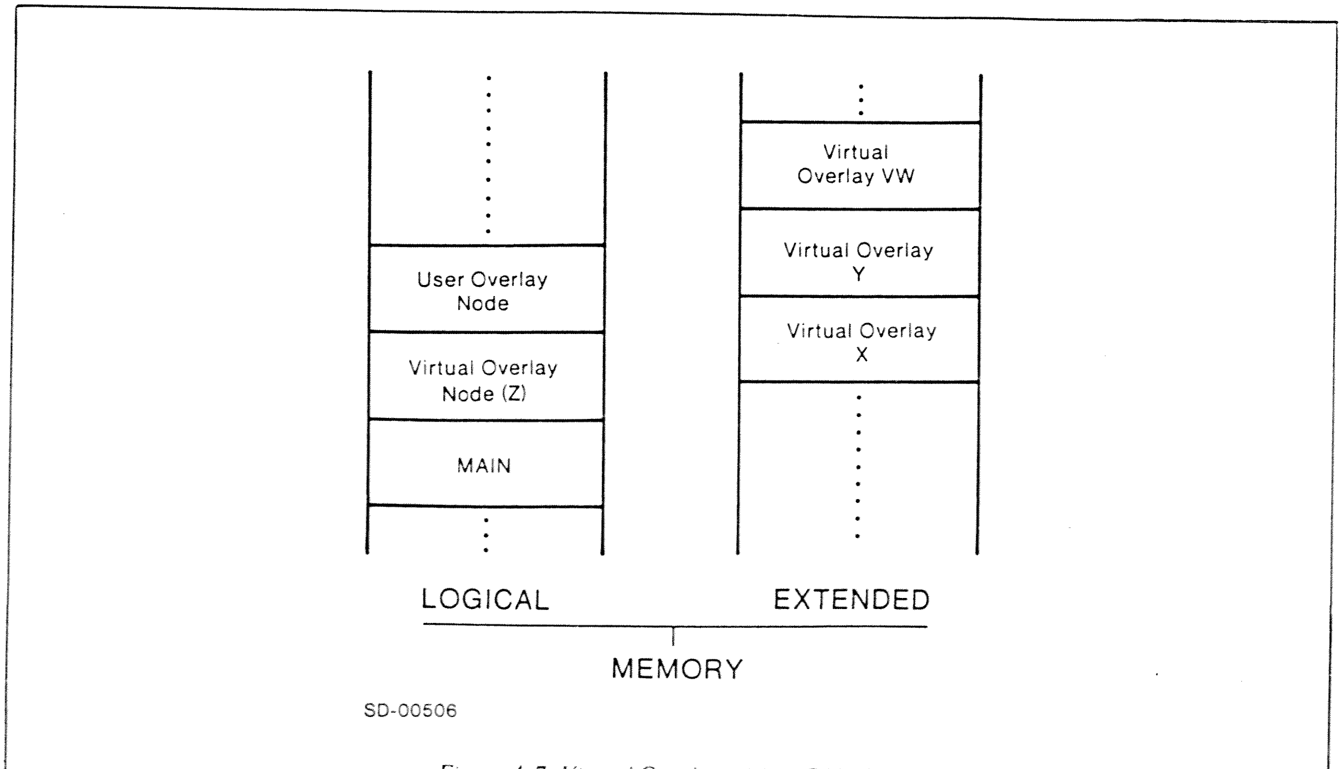


Figure 4-7. Virtual Overlays After .OVLOD

Window Mapping

Swaps, chains, and overlays will help you write a large program which will run in limited address space. If your main program requires more logical memory than the computer has available, you can use another mapped feature: window mapping. Window mapping allows you to gain direct access to portions of extended memory; it also allows you to transfer blocks between extended memory and disk. You can use both virtual overlays and window mapping in one program. Follow these steps to define and use a window map:

1) Determine the amount of memory available for extended addressing use (.VMEM). Do this after all .MEMIs and after .OVOPN.

2) Define the size and position of the window in user address space, and the number of blocks in the extended map (.MAPDF).

3) Logically transfer data between the window and extended memory by activating the memory management unit (task call .REMAP). (Note that no true data transfer occurs; a remap operation changes the *addresses* of the data).

After your program has defined the map, it can repeat .REMAP as often as it wants. It shouldn't issue .REMAP when another task is using the window for I/O; but it can issue extended read/write block calls .ERDB and .EWRB. If it issues other calls, the other task will mistakenly access the new window.

A program can also redefine the window, but a program can have only one window and one window map at a time.

You define windows, like virtual overlays, in multiples of 1024-word pages; they are also page-aligned. Your program accesses data in extended space by redefining the start of the window in logical address space.

RDOS returns blocks allocated to the window via .MAPDF to the pool only when a program executes either a program swap or chain, or a return (.EXEC or .RTN/.ERTN). If your program performs a swap, the window goes away, and the program must redefine it. Note that after a .BREAK call or trap, the state of the window in the break file is indeterminate.

Determine the Number of Free Blocks (.VMEM)

The CLI command SMEM allocates your address space. System call .VMEM provides you with a count of the number of free blocks available to your program for extended map use. If too few blocks are free for your program, you can change memory allotments via the CLI command SMEM.

Required input

none.

AC0 returns the number or free memory blocks for this program.

Format

```
.SYSTM
.VMEM
error return
normal return
```

Possible errors

none in a mapped system.

Define a Window and Window Map (.MAPDF)

As described earlier, window mapping allows your program to transfer data between a window area within logical address space and a series of blocks in extended address space. An extended or window map contains a list of physical memory blocks which can be mapped into the window. System call .MAPDF defines a window and window map; only one window and map can exist within a program. You must define the window area in the address space below NMAX.

The .MAPDF call will assign relative extended block numbers 0 to n-1 to the blocks in extended memory; n equals the number specified in AC0. The first window block in logical space will receive *extended* relative block number 0, the second block (if any) in logical space will receive number 1; the numbers proceed sequentially in extended space. (See Figure 4-8.) Note that defining the window map will not alter the initial contents of the window.

Required input

AC0 - Total number of memory blocks to be assigned to the extended memory area. (This number includes those blocks in logical address space which currently reside within the window.)

AC1 - The starting page number for the window in logical space, from 1 through 31 (decimal) (30 for NOVA 830s and 840s). You cannot specify the first block, (block number 0) because it includes page zero. Also, since the window is block-aligned, remember that the logical address of the beginning of the window falls at the start of a block.

AC2 - The size of the window in 1K blocks.

Format

```
.SYSTM
.MAPDF
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
26	ERMEM	The specified block is out of the range of the window or the window map.

Example

Assume that you want to define a window of 2K in logical space, with 10 blocks total in extended address space. Given the rest of your program, you want to start the window at 20000_8; it will end at 23777_8. This sequence defines the map you want:

```
        .
        .SYSTM
        .VMEM       ;ALWAYS CHECK THE NUMBER
        .           ;OF     EXTENDED      BLOCKS
                    AVAILABLE.
        .           ;(THIS CODE GIVES THE PROGRAM
                    AN OPTION, IF,
        .           ;FOR  WHATEVER  REASON,  THE
                    REQUIRED
                    ;NUMBER   OF   16   1K   BLOCKS
                    AREN'T AVAILABLE.)
LDA 0,C10   ;TOTAL SIZE OF WINDOW (2
            ;BLOCKS IN LOGICAL SPACE,
            ;10 TOTAL IN EXTENDED
            ;SPACE).
LDA 1,C8    ;BOTTOM OF WINDOW AT 20000=
            ;RELATIVE LOGICAL BLOCK
            ;SPECIFY 2 BLOCKS
LDA 2,C2    ;IN AC2.
.SYSTM
.MAPDF      ;DEFINE THE MAP.
JMP ER
  .
  .
  .

C10: 10.
C8: 8.
C2: 2
```

Figure 4-8 shows what logical and extended memory will look like after this sequence:

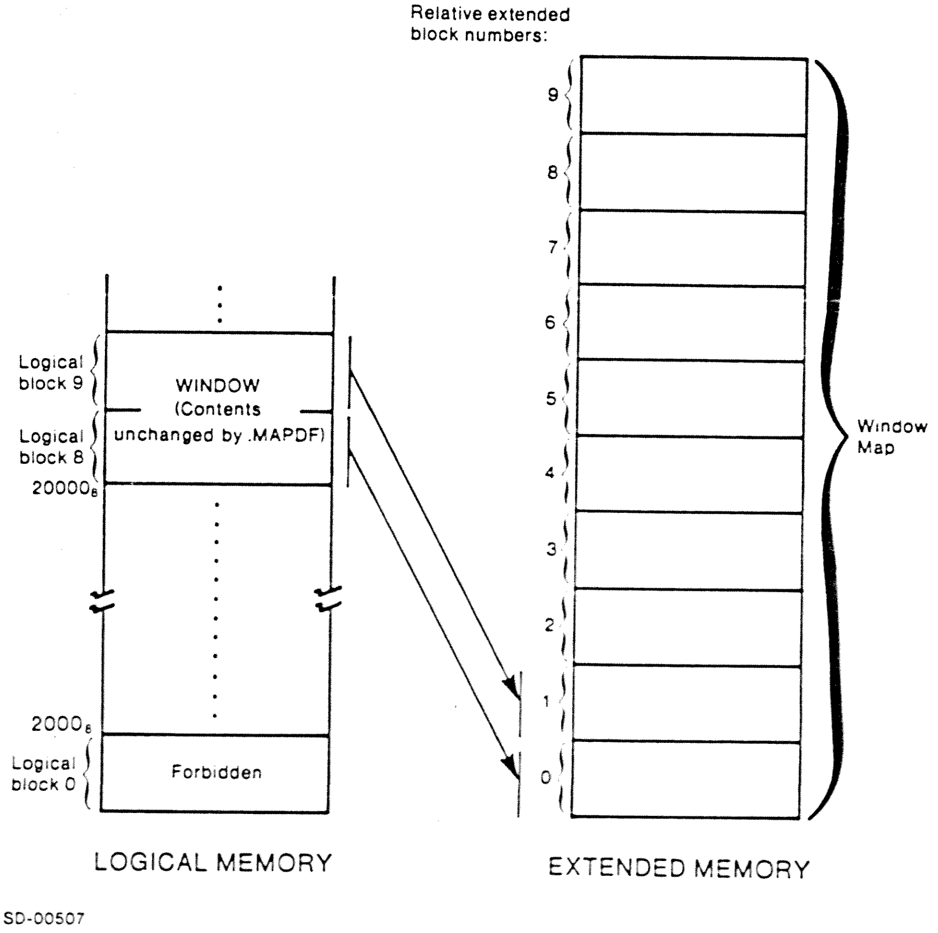


Figure 4-8. Defining a Window Map

Activate a Logical Window Transfer (.REMAP)

Once your program has defined a window and window map, it can remap data from the memory in the window map to or from any part of the window in logical space. The .REMAP call performs a remap operation by placing blocks from the extended address area into the window. An example sequence of mapping operations will illustrate the use of .REMAP.

Note that .REMAP is a task call; hence you must specify the name .REMAP in an .EXTN statement.

Required input

AC1 - Left byte: Starting relative block number in the map (extended space). If you have an array processor, pass the starting relative block number of the array processor. Right byte: Starting relative block number in the window.

AC2 - Number of blocks you wish to remap to the window area. If you have an array processor, pass the number of blocks in AP memory, in two's complement.

Note that block numbers within the windows and the map are relative numbers beginning with 0 (see Figure 4-9, below).

Format

```
.REMAP
error return
normal return
```

The contents of all accumulators are lost upon return from this call.

Possible error

AC2	Mnemonic	Meaning
32	ERADR	Illegal starting address.

We have defined (under .MAPDF) the two-block window and ten-block window map shown in Figure 4-9. The blocks currently in the window have become relative block numbers 0 and 1. The program now performs a remap with block numbers 2 and 3 of the map. RDOS maps blocks 2 and 3 from the extended address area into the logical window. The remap occurs with little system overhead, since RDOS does not actually transfer data between memory locations; it merely updates the map of the memory management unit and then triggers that map. As we mentioned earlier, your programs shouldn't issue a .REMAP when another task has I/O outstanding to or from the window. This would cause the other task's I/O to reference the new window.

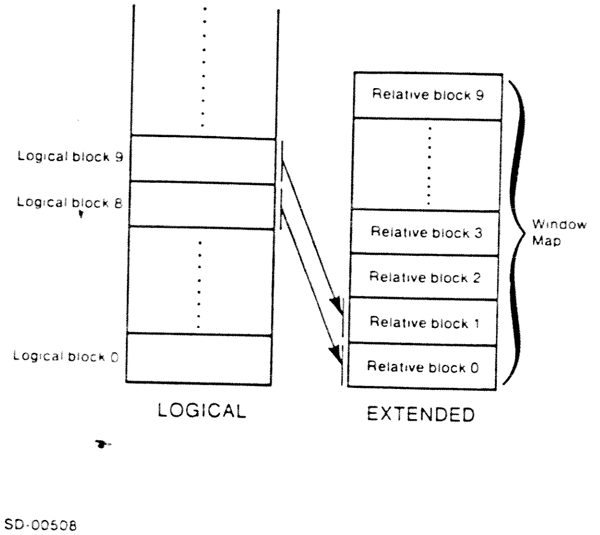


Figure 4-9. Memory before Remap

```
.EXTN .REMAP
      .
      .       ;THE CODE IN FIGURE 4-8 IS
      .       ;IN HERE.
      .
LDA 1,BLK2;PUT 1ST BLOCK NUMBER(S)
          ;TO BE REMAPPED IN LEFT BYTE
          ;OF AC1.
          ;PUT 1ST BLOCK IN WINDOW INTO
          ;RIGHT BYTE OF AC1.
          ;AC1 NOW CONTAINS THE
          ;CORRECT DATA
          ;IN EACH BYTE FOR THE REMAP.
LDA 2,C2  ;SPECIFY THE NUMBER OF
          ;BLOCKS TO
          ;BE REMAPPED IN AC2 (2).
.REMAP    ;DO IT!
      .
      .
      .

BLK 2: 2B7-0B15
C2: 2
```

This sequence remaps 2 blocks (relative block numbers 2 and 3) into the window. In this sequence, we have mapped both blocks, but this wasn't required; we could have mapped either of the blocks independently. Figure 4-10 shows the results of the remap.

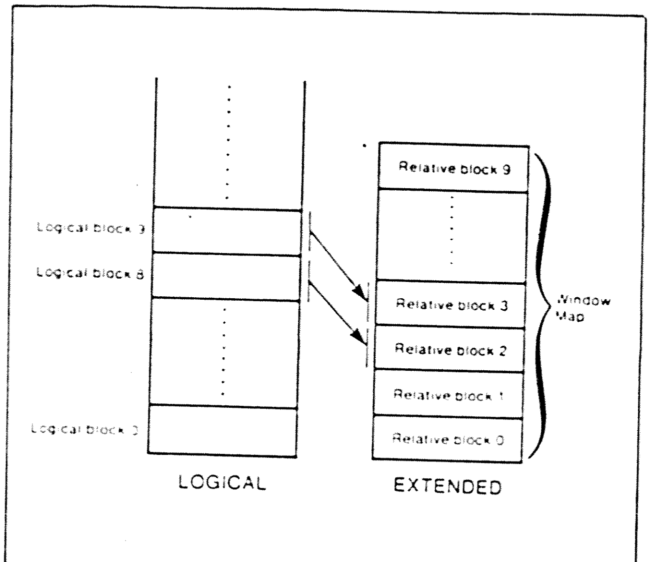


Figure 4-10. Remapping

Extended Direct-Block I/O

After your program has defined a window map, it can use extended direct-block I/O. This special form of I/O is similar in concept and in operation to direct-block I/O, as described in Chapter 3 (see .RDB/.WRB). Direct block I/O, you will recall, transfers 256-word blocks between core and disk, without using a intermediary system buffer.

The extended direct-block I/O calls -..ERDB and .EWRB - can transfer 256-word data blocks between the map in extended memory and disk files. This I/O type provides a quick means of altering data in the extended memory area. Your reference is independent of any remaps which have occurred or may occur during execution of these calls. Moreover, it can transfer disk file data directly to the extended memory area, without passing it through the window in logical address space. Neither .ERDB or .EWRB use an intermediary buffer. The calling sequence of .ERDB and .EWRB resembles the direct-block calling sequence. These are described below.

Extended Direct Block Read (.ERDB)

System call .ERDB can read one or more 256-word disk blocks from a randomly- or contiguously-organized file into one or more 1024- word extended memory blocks.

This call resembles .RDB. However, since RDOS uses the map in extended memory instead of directly-addressable logical memory, the parameter you pass in AC0 to .ERDB differs from that passed to .RDB. (See the example below.) The parameter you pass in AC0 to .ERDB specifies both the map's relative memory block number (in the range 0-244) and a 256-word offset into this block. Since you must have defined a window map (.MAPDF) to use this call, you should know the relative block numbers in the map.

Required input

AC0 - Right byte: extended memory block number in map.
Left byte: 0, read into first 256-word group. 1, read into second 256-word group. 2, read into third 256-word group. 3, read into fourth 256-word group.

AC1 - Starting relative disk block number in the file: from 0 to *n* -1 for a file consisting of *n* disk blocks.

AC2 - Left byte: number of 256-word disk blocks to be read.
Right byte: channel number (if file was opened on channel 77).

Format

```
.SYSTM
.ERDBn              ;Read from the disk file
                    ;opened on channel n
                    ;(or 77).
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
4	ERSVI	Not a randomly- or contiguously-organized file.
6	EREOF*	End of file.
7	ERRPR	File is read-protected.
15	ERFOP	No file is open on this channel.
30	ERFIL	File read error (mag tape or cassette, bad tape).
40	EROVA	File not accessible by direct-block I/O.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

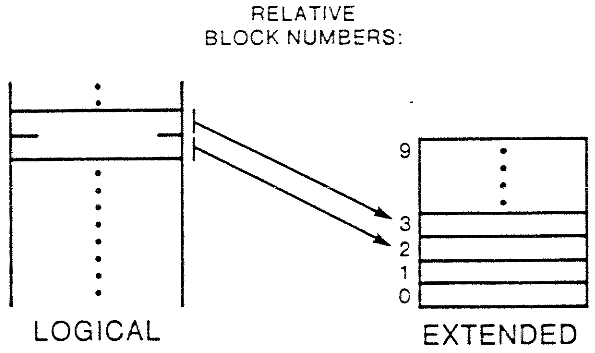
*Upon detection of error EREOF, RDOS returns the code in the right byte of AC2; the left byte contains the partial read count.

Example

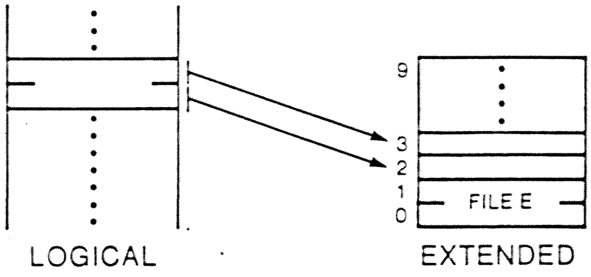
You can use the following code to transfer disk file E to the map via .ERDB. Let's continue the configuration we showed in Figure 4-10. The code in Figure 4-11 writes file E to relative block numbers 0 and 1 in the map.

task can still access extended blocks 2 and 3.

The 256-word offset into the selected block will indicate either 0, 400_8, 1000_8, or 1400_8 for the start of each 256-word disk block. RDOS adds the extended memory block number and the offset. This produces the memory block number in the right byte and either 0, 1, 2, 3 in the left byte for the first, second, third, or fourth 256-word disk block.



```
 LDA 0, FILEE        ;BYTE POINTER TO DISK FILE E.
 SUB 1, 1            ;DEFAULT DISABLE MASK.
   .SYSTM
   .OPEN 3           ;OPEN E ON CHANNEL 3.
    •
    •
    •                ;
   SUB 0, 0          ;GET 0 TO START READING
                     ;TO EXTENDED BLOCK 0.
   SUB 1, 1          ;GET 0 TO START READING
                     ;FROM STARTING POSITION IN
                     ;DISK FILE E.
   LDA 2, C8         ;SPECIFY THE NUMBER OF DISK
                     ;BLOCKS TO BE READ, IN LEFT
                     ;BYTE OF AC2 -- 8. THESE
                     ;BLOCKS WILL FILL MAP BLOCKS
                     ;0 AND 1.
  .SYSTM
  .ERDB 3            ;READ FROM FILE E ON CHANNEL 3.
    •
    •
    •
    •
   C8: 8.B7
FILEE:.x1*2
        .TXT "E"
```



SD00746

Figure 4-10. Extended Block Read.

Extended Direct Block Write (.EWRB)

Use .EWRB to write one or more 256-word blocks from extended memory to a randomly- or contiguously-organized disk file. The current contents of the window remain unchanged, as do the contents of the map.

This call resembles .WRB, but the parameter passed in AC0 to .EWRB differs from that passed to .WRB. AC0 must specify both the relative extended memory block number (in the range 0-244) and a 256-word offset into this block. Your program must have defined a map via .MAPDF to use this call; hence you should know the relative 1K block numbers in the map. See .ERDB for details on the offset.

Required input

AC0 - Right byte: extended memory block number
Left byte: 0, write from first 256-word group 1, write from second 256-word group 2, write from third 256-word group 3, write from fourth 256-word group

AC1 - Start writing to this relative block number in the disk file.

AC2 - Left byte: number of 256-word blocks to be written.
Right byte: channel number (if file was opened on channel 77).

Format

```
.SYSTM
.EWRB n          ;Write to the disk
                 ;file opened on
                 ;channel n (or 77).
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
4	ERSVI	Not a randomly-or contiguously-organized file.
6	EREOF	End of file in a contiguous file.
10	ERWPR	File is write-protected.
15	ERFOP	No file is opened on this channel.
27	ERSPC*	Disk space is exhausted.
40	EROVA	File not accessible by direct-block I/O.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

*Upon detection of error ERSPC, RDOS returns the code in the right byte of AC2; the left byte contains the partial write count.

End of Chapter

Chapter 5
Multitask Programming

This chapter describes tasks, task management, task overlay management, and task control from the console via operator messages. It begins by explaining task prioritites and the Task Control Block (which the RDOS Task Scheduler uses to keep track of each task in a program), it then describes the possible task states, and the User Status Table, which monitors all TCBs during program execution. Finally it gives the calls which a task can issue to control itself or other tasks. Your program is the initial task; after it initiates one or more tasks, any of those tasks can issue a task or system call.

After you have written a task routine, you should assign a task ID to it. An ID number is not mandatory, but certain useful task calls, as well as the OPCOM console communications feature, require it. Your program then proceeds to initiate the task via .TASK or .QTSK (or, from the console, via OPCOM commands RUN or QUE); RDOS then assigns the task a TCB from the TCB pool you established either via a .COMM TASK statement or during loading. The task is then *ready* for execution. Depending on its priority, and other conditions specified in your program, the task achieves CPU control and executes. It retains CPU control until it either suspends itself, or is suspended by an equal or higher priority task's request for the CPU after an interrupt. The task's TCB saves its current state. The program's User Status Table monitors all TCBs and their associated tasks; this enables the Task Scheduler to resume execution of any suspended task from the point of suspension. The task retains its TCB until the task kills itself, or is killed, via .KILL, .AKILL, .ABORT, .OVKIL, or OPCOM command KIL. After a task is killed, its TCB returns to the free TCB pool and the task remains inert until you re-initiate it, and it receives another TCB.

Each task you write into your program is memory-resident during program execution, unless it resides in an overlay. If it resides in an overlay, your program must use .OVOPN, then load the overlay via either .TOVLD or .QTSK.

The following list summarizes the major headings in this chapter, and the calls these sections contain.

Task Initiation

.TASK Create a task with the specified priority and ID number. Your program must issue .TASK or .QTSK to initiate a multitask environment.

Task Termination

.KILAD Pass control to this address when a task defining a kill-processing address is killed.

.KILL Kill the calling task (i.e., kill yourself).

.AKILL Kill all tasks of the specified priority.

.ABORT Kill the specified task and its currently-executing system call (if any).

Task State Modification

.PRI Change the priority of the calling task (yourself).

.ARDY Ready all tasks of a given priority.

.SUSP Suspend the calling task (yourself).

.ASUSP Suspend all tasks of a given priority.

Intertask Communication

.XMT Transmit a one-word message to a given address for eventual receipt by another task.

.XMTW Transmit a one-word message (as in .XMT) and wait (suspend yourself) until the other task receives the message.

.IXMT Transmit a one-word message (as in .XMT) from a special (nonstandard) user-defined interrupt routine.

.REC Receive a message from another task.

Overlay Management

.TOVLD Load the overlays which were opened on channel n, for either your own, or another task's, use.

.OVREL Release an overlay.

.OVEX Release an overlay and return to a specified address.

.OVKIL Kill the caller (yourself) and release its overlay node.

Enqueuing Tasks

.QTSK	Create a task at the specified priority; place it in the task queue for execution at the specified time, and execute a specified number of times.
.DQTSK	Dequeue a task which has been enqueued by .QTSK.

User System Clock Commands

.DELAY	Delay the caller (yourself) for the specified number of RTC pulses.
.DUCLK	Define a periodic interrupt, at which system control will go to the specified address.
.UCEX	Return to the system after executing a routine specified in the .DUCLK address.
.RUCLK	Remove the interrupt interval and address specified in .DUCLK from the system.
.GHRZ	Get the frequency of the real-time clock.

Managing Tasks by ID Number

.IDST	Get a task's status.
.TIDP	Change a task's priority.
.TIDR	Ready a task.
.TIDS	Suspend a task.
.TIDK	Kill a task.

Task/Operator Communications Calls

.TWROP	Write a message to the console.
.TRDOP	Read a message from the console.

Task/Operator Module (OPCOM) Commands

DEQ	Dequeue a QUEued task.
KIL	Kill a task.
PRI	Change a task's priority.
QUE	Queue a task for periodic execution.
RDY	Ready a task.
RUN	Execute a task.
SUS	Suspend a task.
TST	Display a task's status.

Disabling the Multitask Environment or Task Scheduler

.SINGL	Disable the multitask environment.
.MULTI	Enable the multitask environment.
.DRSCH	Disable the task scheduler.
.ERSCH	Re-enable the scheduler.

Task Priorities

Task priorities range from 0 (highest priority) through 255 decimal. RDOS automatically creates one task at priority 0 for the task whose starting address you specify in the .END statement at the end of your program.

Several tasks may exist at the same priority. Equal priority tasks receive CPU control on a round-robin basis, which means that the task which most recently received control will be the last to receive control again, unless other tasks are unable to receive control at the moment that rescheduling occurs. Whenever your program changes a task's priority (.PRI), RDOS places the task at the end of the list of all tasks within its new priority.

Task Control Blocks

A task is an asynchronous execution path through user address space which demands the use of system resources. You can assign many tasks to a single reentrant path, and you can assign each of these tasks a unique priority. Given the asynchronous nature of tasks, the RDOS Task Scheduler must maintain certain status information about each task. RDOS retains this information within a Task Control Block (TCB); there is one TCB for each task. The following illustration describes the structure of TCBs:

Word	Mnemonic	Contents
0	TPC	User PC (B0-14) and Carry (B15).
1	TAC0	AC0.
2	TAC1	AC1.
3	TAC2	AC2.
4	TAC3	AC3.
5	TPRST	Status bits and priority.
6	TSYS	System call word.
7	TLNK	Link word, to next TCB.
10	TUSP	USP (User Stack Pointer).
11	TELN	Extended save area.
12	TID	Task ID number, right byte.
13	TTMP	Scheduler temporary storage.
14	TKLAD	Task kill address (if program specified one).
15	TSP	Stack pointer.
16	TFP	Frame pointer.
17	TSL	Stack limit.
20	TSO	Overflow address (single task environment).

Words 1-4 in the TCB are self-explanatory. Word 5, TPRST, contains the task state and priority information shown in Figure 5-1.

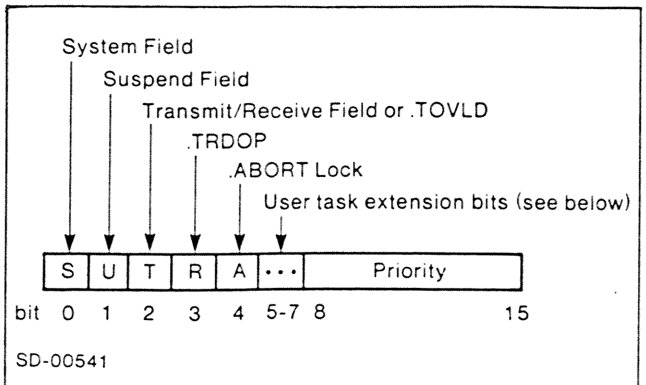


Figure 5-1. Task State/Priority Information (TPRST)

The Task Scheduler sets the fields in TPRST as follows:

Field	Bit Setting/Meaning
S	1 = Task has issued a system call and has been suspended until the call is done. 0 = System call is done, or no call is outstanding for the task.
U	1 = Task is suspended by .SUSP, .ASUSP, or .TIDS.
T	1 = Task has issued either .XMTW/.REC or .TOVLD.
R	1 = Task is awaiting a message via .TRDOP.
A	1 = Task is being aborted.

Bit 5 is reserved; bits 6 and 7 allow you to expand the RDOS task-handling mechanism, as described in Appendix J.

Bits 8-15 contain the task priority.

RDOS uses TSYS, word 6, to store information about system calls and in .XMTW/.REC/.TOVLD.

TLNK contains the starting address of the next TCB in the chain. TUSP contains the value of location USP at the time this task last changed from the executing state. You may use USP as a general-purpose storage location for each task while the task is executing. The system will restore the USP value for each task that gains CPU control.

TELN points to the task's higher-level language save area; if you do not use it, the system sets TELN to 0.

TID contains the task identification number, if any, in its right byte.

TKLAD, word 14, contains the address which is to receive control whenever a task is killed, if you have defined such an address via a .KILAD call. Bit 0 is set if a .KILL or .ABORT of the task has been issued.

The remaining four words contain stack state save information which is reserved for TCBs.

Building Multitask Programs

Before you run a multitask program, you must specify both the number of RDOS channels and the number of TCBs which that program needs. You can do this before assembly, within the program, via a .COMM TASK statement. You can also specify tasks and channels with the /K and /C local switches in the RLDR command line. If you use .COMM TASK, it must appear in your first binary in the RLDR command line, since it affects the loading process of the remainder of the program and determines which task scheduler (TMIN or TCBMON) will become part of the program. If you use either the /C or /K switch along with a .COMM TASK statement, the switch information overrides the statement specification. The format of the source program statements is:

```
.COMM TASK, k*400+c
```

where:

k represents the octal number of tasks and
c represents the octal number of RDOS channels which your program will use.

Example:

```
.COMM TASK, 7*400+16
```

In mapped systems, the maximum number of tasks (k) cannot exceed 44_{10}. This is due to the requirement (in mapped systems) that all TCBs must reside in NREL, in the first 1K page of memory. If the program uses overlays, the overlay directory must also reside in the first 1K page, which reduces space for TCBs.

Data General supplied TMIN and TCBMON, all task command modules, the interrupt-on symbolic debugger, and BFPKG (see *Application Note: Buffered I/O in RDOS)* in the system library (SYS.LB). Unless you specify otherwise with an RLDR switch, RLDR will place all items required from the library directly above the program code.

Note: Because the system library (SYS.LB) differs for each type of system (e.g. unmapped NOVA and mapped NOVA), programs loaded under one type of system will probably not execute under another type of system. To load for a different kind of system, you must obtain the proper system library for the target system, and ensure the RLDR searches it, not the current library, during the load. You can do this by loading from a subdirectory which contains the target system library and links to RLDR.

If you wish to write your own task command modules, or define a task memory or FPU save area, see the source listings for the system library (if you acquired library source listings with your system).

Conserving ZREL Space

Normally, each different task call a program will use requires one word of page zero (ZREL) space. For example, in the following conventional use of call .TASK:

```
.EXTN .TASK
 .
 .            ;Set up Accumulators.
.TASK
 .
```

The task call word .TASK is resolved by SYS.LB to a JSR instruction which transfers control through a page zero address. Thus .TASK requires one word of ZREL (other subsequent .TASK calls will not require additional ZREL). If you want to conserve on ZREL space, you can use an alternate method. Replace each task call with a transfer to a label which has the same name as the original task call, but with the first two characters transposed. The transfer must be a JSR or equivalent. You must declare the transposed call .EXTN. For example:

```
        .EXTN T.ASK
        .
        .      ;Set up accumulators.
        JSR @TASKO
        TASKO: T.ASK
```

The transposition scheme uses no ZREL space.

Task States

A task can exist in any of three states. Tasks are either *ready* to perform their functions, or they are actually in control of the CPU and are *executing* their assigned instruction paths, or they are *suspended* and temporarily unable to receive CPU control. A task can also be *dormant*, having relinquished its TCB (or never having had a TCB); a dormant task has no priority and no chance of gaining CPU control until activated by a .TASK or .QTSK command.

The Task Scheduler always gives CPU control to the highest priority task that is *ready*.

Suspended tasks are tasks which have at least one of the four status bits (S, U, T, R) in TPRST set to 1. A task may become suspended for one or more of the following reasons:

1. It has been suspended by .ASUSP or .TIDS.
2. It has suspended itself for a specified period by .DELAY, or an indefinite period via .SUSP.
3. It is waiting for a message from another task, .REC.
4. It has issued a message-and-wait call, .XMTW.
5. It is waiting for the use of an overlay node.
6. It has issued a .SYSTM call, and is waiting for completion of that call.

Just as a number of different events can suspend a ready task, several events can ready a suspended task:

1. The action of .ARDY or .TIDR task calls on the task.
2. The receipt of a message by a task which has issued .REC.
3. The loading of a requested overlay.
4. The completion of a .SYSTM call (such as a request for I/O).

If a task is suspended by both a task suspend call and by some other event, you must ready the task both by an .ARDY (or .TIDR) call and by whatever other event suspended it. Thus a task may be doubly suspended, with both bits S and U set in the task's priority and status word, TPRST. The environment must allow RDOS to reset bits S, U, T and R to ready the task.

You can delete tasks from the active queue and place them in dormancy, either separately (.KILL, .TIDK or .ABORT) or by priority group (.AKILL). Tasks which you have deleted add their empty TCBs to an inactive chain of free element TCBs.

If all tasks are killed, and no task is awaiting execution via .QTSK, the effect is the same as:

```
.SYSTM
.RTN
```

Program control then returns to the next higher program level.

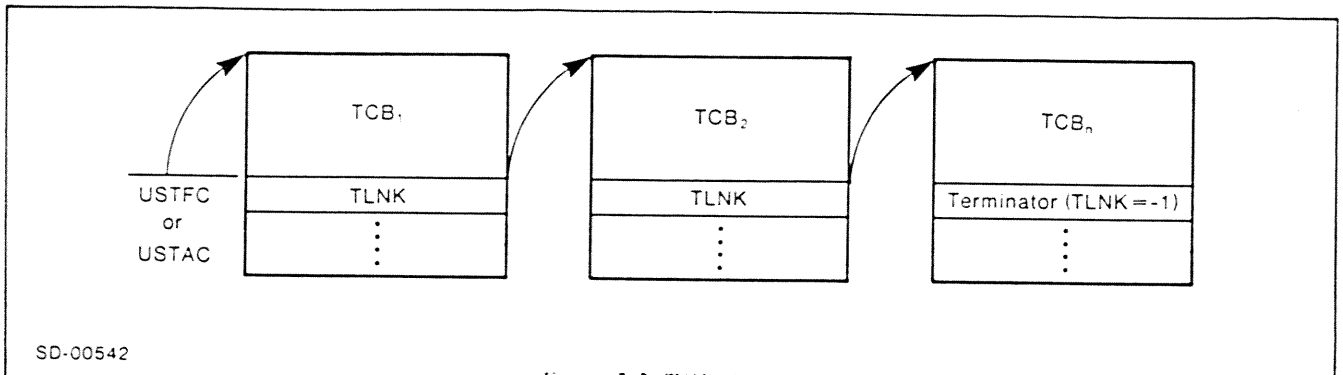


Figure 5-2. TCB Chain

TCB Queues

There is one TCB queue for tasks which are currently executing, suspended, or ready. This queue consists of a chain of TCBs, connected by the TLNK word in each TCB, and is called the active chain. USTAC of the User Status Table points to the first TCB. This TCB points to the next TCB, etc. The last TCB in the chain has a TLNK of -1.

The free element TCB chain is a simple queue of dormant TCBs. TCBs in the free element chain are joined by TLNK words; all other words in each of these TCBs are unused. There is no priority among TCBs in the free element chain. USTFC of the User Status Table points to the first TCB in the free element chain (see Figure 5-2).

Task Synchronization and Communication

Each task can communicate with another by sending a one-word message to an agreed-upon location in user address space. This address space includes all locations from address 16 through NMAX. (But avoid locations 0-17_8 and 40_8 -47_8 in ZREL and system tables directly above 400_8.)

The task sending a message may either return to the Task Scheduler immediately (.XMT) or it may suspend itself (.XMTW) until a receiving task has issued a receive request (.REC) and has received the message. Receipt of the message includes the resetting of the contents of the message location to zero. Upon receipt of the message, the recipient task has bit T set to 0. The message location must contain 0 before the message is sent.

User Status Table

The User Status Table (UST) is a 24_8 word table which records runtime information about a program. This table is located at addresses 0400_8 through 0423_8 and has the following structure in memory:

Address	Label	Contents
012	USTP	ZREL pointer to UST.*
400	USTPC	Used by the system.
401	USTZM	ZMAX.
402	USTSS	Start of Symbol Table (SST).
403	USTES	End of Symbol Table (EST).
404	USTNM	NMAX after runtime .MEMIs.
405	USTSA	Starting address of Task Scheduler.
406	USTDA	Debugger address; -1 if the debugger wasn't loaded.
407	USTHU	USTNM after loading (original NMAX).
410	USTCS	FORTRAN common area size.
411	USTIT	CTRL-A interrupt address; -1 initially.
412	USTBR	CTRL-C or .BREAK address; -1 initially.
413	USTCH	Number of TCBs (left byte) and channels (right byte).
414	USTCT	Current TCB pointer.
415	USTAC	Start of active TCB chain.
416	USTFC	Start of free TCB chain.
417	USTIN	Initial start of NREL code (INMAX).
420	USTOD	Overlay directory address.
421	USTSV	Available for use by the system.
422	USTRV	Revision level number, and, during execution, the environment state.
423	USTIA	Address of TCB for console interrupt task; 0 initially.

*The UST for a program running in an unmapped foreground starts at the beginning of the foreground memory partition.

Location 12, USTP, in page zero, points to the start of the UST belonging to the currently executing foreground or background program. The loader creates symbol USTAD as an .ENTry; USTAD also points to the base of the program's UST.

USTPC indicates which program is running; 0 indicates the background is running, and 1 indicates that the foreground is running; it also provides compability with SOS.

USTZM contains ZMAX, the first free location in page zero after loading.

Locations 402 and 403, USTSS and USTES, point to the start and end of the symbol table, respectively. By default, the loader loads the symbol table so that one location after the last location in the symbol table coincides with the value of NMAX. If you request that RDOS place the symbol table in upper memory (by using the global /S switch in the RLDR command), the symbol table is moved so that it will be immediately below RDOS space when the save file is executed. If the symbol table has not been loaded, locations 402 and 403 contain zeroes.

USTNM contains the current value of NMAX at run time. This value changes as NMAX is increased or decreased. Location 407, USTHU, is set by the loader to the value of NMAX after loading. RDOS never changes this word during program execution.

USTIT is the interrupt address (CTRL A). After loading, this address is set to -1. If it is unchanged at run time, control goes to the next higher level program with USTIT set to a valid address when a CTRL A interrupt occurs. (If the foreground is interrupted and there is no higher level program in the foreground with a valid USTIT address, RDOS terminates the foreground.) The user core image is not saved. Your program can set USTIT an execution time to an address to which the system will transfer control on a CTRL A interrupt.

USTBR is the break address (CTRL C). After loading, RDOS sets this address to -1. Whenever a CTRL C break occurs, the system will write the core image to file BREAK.SV (or FBREAK.SV in the foreground) in the current directory. If USTBR remains unchanged at run time, control goes to the next higher level program with USTBR set to a valid address when a CTRL C interrupt occurs. Alternatively, you can set USTBR to an address to which control will be directed upon the successful creation of the break file. If RDOS can't create the break file (e.g., because it is out of disk space), control will go to the address specified by (USTBR) -1, i.e., one less than the address contained in USTBR; AC2 will contain the error code.

USTCH contains the number of program TCBs in its left byte, and the number of I/O channels in the right byte.

USTSV is reserved for RDOS.

USTRV is reserved for storage of the revision number information for this save file, and for runtime data on the machine which is running the program. Revision numbers can extend from 00 to 256; RDOS stores the major revision number in the left byte, and the minor revision number in the right byte of this word. While a program is running, USTRV contains values which indicate what kind of machine and RDOS system is running the program. You can find these values and interpretations under ENVIRONMENT STATUS BITS IN USTRV, in file PAR U.SR in Appendix B.

USTIA contains the TCB address of the task that issued an .INTAD system command. The loader initializes this word to 0.

Task and System Calls

There are four essential differences between task calls and system calls: first, task calls have no .SYSTM mechanism; each call uses a module from the system library, and therefore *you must declare each call included in a program as external, in an .EXTN statement.* If your program doesn't declare each call external, the loader won't load the call's module, and the call won't work.

The second difference relates to the first: RDOS executes all system calls in *RDOS* space, but executes all tasks calls in *user* space. Therefore the diversity of *task* calls in a program affects the program size, whereas the diversity of *system* calls does not.

The third difference is that most task calls do not have error returns, and hence do not reserve an error return location.

The fourth difference is that you use the accumulators to pass all parameters to most task calls. You will generally use AC0 and AC1 to enter or return data. You occasionally use AC2 to enter data; when an error is defined for a call, AC2 will contain the code on an error return.

By default, on return from all task calls, AC3 will contain USP, the contents of location 16_8. RDOS maintains the frame pointer in location CSP. If you have a NOVA 3, you can return the contents of the hardware frame pointer in AC3 by loading the program with module N3SAC. (In NOVA 3s, the hardware stack is moved for each task swap, but the stack overflow handler remains at location 43_8. On an ECLIPSE, you can return the frame pointer by inserting module ESAC3 in the loader command line. Here is a summary of returns in AC3:

If program was loaded with module:	Then upon return from call AC3 contains contents of:
NSAC3 (any machine; always used by default)	USP (location 16_8.
NSAC3 (NOVA 3s only)	Frame pointer register.
ESAC3 (ECLIPSEs only)	Frame pointer (location 41_8).

In summary, task calls differ from .SYSTM calls in four ways:

1. Task calls reference library modules, and must be declared external. Task calls are not preceded by the .SYSTM mnemonic, and are resolved by the binder/loader to be *JSR* calls to task processing modules.

2. Task calls are processed in user address space, while RDOS or system calls require system action which occurs in RDOS space.

3. Not all task calls have error returns. Those which do not have error returns do not reserve an error return location.

4. You must pass all parameters to task calls via the accumulators (except .QTSK).

Task Initiation

The .TASK command will initiate any memory-resident task. The .QTSK command, described under ENQUEUING TASKS, will initiate either a core-resident or overlayed task for periodic execution.

Create a Task (.TASK)

This command initiates a new task at a specified priority in your program, and assigns an identificatiion number to the task, if you desire. When you load the program, only one task exists; therefore your system must issue this call (or .QTSK) to initiate a multitask environment.

.TASK will pass the contents of AC2 to the created task. This permits your program to relay an initial one-word message to the newly-created task.

Required input

AC0 - Right byte: priority of the new task (range: 1 to 377). If you set this byte to zero, the priority of the new task will be identical to the calling task's priority. Left byte: (optional but recommended) ID number for the new task (range 1 to 377). You may give an ID number of zero to more than one task. Each nonzero ID must be unique.

AC1 - Address where the new task will begin execution.

Format

```
.TASK
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
42	ERNOT	No TCBs available.
61	ERTID	A task with the requested ID (except 0) already exists.

Note: Error codes listed under each call represent the most common errors, and the meanings have been expanded and interpreted in light of the call.

Task Termination

This section and the following sections describe the calls your program can use to manipulate tasks without using their ID numbers. To control tasks by ID number, see "Managing Tasks by ID Number," later in this chapter.

Your program can kill a group of tasks by priority (.AKILL), or an individual task by ID number (.TIDK (described later) or .ABORT), or it can instruct a task to kill itself (.KILL). The .KILL call has no return.

To allow your program to proceed efficiently, RDOS provides the .KILAD call. This specifies an address to receive control before a task is killed. The .KILAD address can instruct the task to close its channel(s), release its overlay(s), or give it a choice of action.

Upon most orderly task terminations, .AKILL or .TIDK, RDOS raises each task you are terminating to the highest possible priority and readies it. If several tasks exist with a priority of 0, RDOS will service these tasks before killing the specified task(s). Thus if a task was suspended by a .REC, .XMTW, .SUSP, or .TIDS task call, RDOS would lift the suspension. If the task were in suspension because it had a .SYSTM call outstanding, RDOS would complete that call before readying the task. In either case, RDOS terminates the task you wish to kill when the task receives CPU control, unless your program has specified a kill-processing address.

If you specify a kill-processing address via .KILAD, control will go to that address when the task achieves CPU control. This will allow the task to close any channels or release (.OVREL) any overlays it was using. Moreover, the kill-processing routine can act as a reprieve, since RDOS will not actually terminate the task processing routine until it is killed a second time. The kill-processing routine can thus act as a validation procedure to determine whether or not the target task should be terminated. At this point, the task being killed can renew its kill-processing address by re-issuing .KILAD.

After a task has been killed by any means, it relinquishes its TCB to the free TCB pool for possible use by future tasks.

Define a Kill-Processing Address (.KILAD)

The .KILAD task call permits a task to define a special address which will gain control the first time that your program tries to terminate the "target task". On your second attempt to kill the task, RDOS will terminate the task without transferring control to the kill-processing address.

The kill address allows a task to release system resources before terminating. Each task must explicitly release such resources as overlays, channels, user devices and user clock definitions -- and you can write code for this into the task's .KILAD routine. After releasing these resources and following any other instructions, the task must then itself issue a .KILL call to terminate itself. On this *second* attempt to terminate the task, termination will occur immediately.

If, on the other hand, the target task decides not to terminate itself, then, before branching out of the kill-processing routine, it should issue a .KILAD call to the same or to a different kill-processing routine. This will ensure that if an attempt is made later to kill this task, it will not be killed immediately but will branch again to its kill-processing routine.

A task in a kill-processing routine is in execution at the highest priority; it has CPU control. Thus such routines *will retain control* until they relinquish this control by a task state transition or by a priority level change.

Required input

AC0 - Address of the kill-processing routine.

Format

```
.KILAD
normal return
```

There are no error returns from this call.

Delete the Calling Task (.KILL)

This command deletes the calling task's TCB from the active queue and places it in the free element TCB chain. The calling task is the only task that you may delete via .KILL. There is no return from this call. If you have defined a kill-processing address for this task, then RDOS raises the task to the highest priority and control returns to the Task Scheduler. Otherwise, control returns to the Task Scheduler so that it can allocate the system resources to the highest priority ready task.

Format

```
.KILL
```

There are no returns from this call.

Kill All Tasks of a Given Priority (.AKILL)

This command first raises all tasks of a given priority to the highest priority, and then either kills them or transfers control to their kill-processing addresses. All TCBs that it deletes from the active queue are placed in the free TCB chain. It also immediately kills any tasks suspended by .XMTW, .TIDS, .REC, or .SUSP. If you attempt to kill a task waiting for completion of a .SYSTM call, RDOS will not delete that task until the .SYSTM call has executed. If the calling task itself belongs to the specified priority, RDOS will delete it.

Required input

AC0 - Priority class of the tasks you wish to kill.

Format

```
.AKILL
normal return
```

There is no error return from this command. If no tasks exist with the priority given in AC0, RDOS takes no action.

Abort a Task (.ABORT)

The .ABORT task call readies a specified task immediately, and makes it execute the equivalent of .KILL call when it gains CPU control. If a .KILL processing address exists, RDOS will transfer control to it. The exact time of completion of the .ABORT depends on the internal priorities of the system. For example, a task attempting to perform a write sequential of 500 bytes might be aborted after writing any number of bytes. You use an ID number to specify the task you want to abort. Thus, the caller can abort either itself or some other ready or suspended task.

Task call .ABORT does not release any open channels used by the aborted task, nor does it release any overlays. All outstanding operations performed by the task, like waiting for a message transmission/reception (.XMTW/.REC), are terminated. Likewise, all *system calls* are aborted, with two exceptions:

1. Calls performing multiplexor or MCA I/O
2. System read or write operator message calls, .RDOPR or .WROPR (Chapter 6).

Your program can abort multiplexor or MCA I/O by closing their channel(s). You can .ABORT operator messages initiated by *task* calls .TRDOP and .TWROP. Messages initiated by the system call versions, .RDOPR and .WROPR, are not aborted. (A single program can't use both task and system versions of these calls.)

Required input

AC1 - ID of the task to be aborted.

Format

```
.ABORT
error return
normal return
```

You will lose the contents of AC0 upon return.

Possible errors

AC2	Mnemonic	Meaning
61	ERTID	An ID of zero was specified, or no such task ID was found.
110	ERABT	The specified task was performing either multiplexor or MCA I/O or a system read/write operator message call; or was being aborted by another task.

Task State Modification

Change the Calling Task's Priority (.PRI)

This command changes the priority of the calling task to the value contained in AC0. RDOS will assign to the calling task the lowest priority in its new priority class; the Task Scheduler will allocate CPU control to all of the other ready tasks in the same class before giving it to this task. Naturally, its position in this priority class will change as rescheduling proceeds.

Required input

AC0 - New priority value for the calling task. If you request a priority higher than 377_8, RDOS will accept only the value in bits 8 through 15.

Format

```
.PRI
normal return
```

There is no error return from this call.

Ready All Tasks of a Given Priority (.ARDY)

This command readies all tasks previously suspended by .ASUSP (.SUSP or .TIDS) whose priority you specify in AC0. That is, this call resets bit U in word TPRST of each TCB that was set by a previous call to .ASUSP, .SUSP, or .TIDS. The system will not ready tasks suspended for additional reasons (e.g., outstanding system calls) until bit S of TPRST in each of the TCBs is also reset (e.g., by receiving a task message via .REC). RDOS cannot ready a task until the program environment allows it to zero both bits S and U of that task's word TPRST.

Required input

AC0 - Priority of task(s) you wish to ready.

Format

```
.ARDY
normal return
```

There is no error return from this command. If there are no tasks with the given priority in AC0, RDOS takes no action.

Suspend the Calling Task (.SUSP)

This command suspends the calling task by setting bit U of that task's TCB to one. The task remains suspended until your program readies it with a .ARDY or .TIDR call.

Format

```
.SUSP
normal return
```

There is no error return.

Suspend all Tasks of a Given Priority (.ASUSP)

This command suspends all tasks of the priority you specify in AC0. The calling task may suspend itself with this call. All tasks suspended by .ASUSP - even those suspended for other reasons (e.g., an outstanding system call, setting bit S of TPRST) - will remain suspended until readied by a .ARDY or .TIDR command.

Required input

AC0 - Priority of the task(s) you wish to suspend.

Format

```
.ASUSP
normal return
```

There is no error return from this command. If no tasks exist with the given priority, RDOS takes no action.

Inter-Task Communication

RDOS provides a mechanism which allows single tasks to transmit and receive one-word messages. You can also use this mechanism to lock a task process and prevent multiple tasks from entering the process concurrently. Your program specifies an address for the one-word message, and it must clear this address to 0 before depositing the message via a transmit call. If several tasks attempt to receive a message from the same address, only the highest priority task will receive the message.

Transmit a Message (.XMT) and Wait (.XMTW)

Each of these calls instructs the calling task to send a one-word nonzero message to an empty (all-zero) message location for another task. If a task has issued a .REC for this location, the task will receive the message and be readied. If no .REC is outstanding, RDOS will deposit the message. .XMTW will not return until the message has been received; .XMT will return as soon as the transmitting task is readied.

Required input

AC0 - The message address in user address space where you want to deposit the message (this address must not have bit 0 set to 1).

AC1 - The one-word, nonzero message which RDOS will pass to the address given in AC0, for receipt by the receiving task.

Format

```
.XMT              or          .XMTW
error return                  error return
normal return                 normal return
```

Possible errors

AC2	Mnemonic	Meaning
43	ERXMT	The message address is already in use.
115	ERXMZ	Zero message word.

Transmit a Message From a User Interrupt Service Routine (.IXMT)

The .IXMT* call enables an interrupt routine to send a message to a task in the current environment. .IXMT from an interrupt routine has the same effect as a .XMT from a task.

Your program can specify a nonSYSGENed device via the call .IDEF (Chapter 7). When such a user-defined device interrupt occurs, control passes to the interrupt service routine which you have written for the device. RDOS freezes the entire task environment while the interrupt routine executes; the routine then ends with task call .IUEX. If AC1 contains 0 at the .UIEX, RDOS will restart the environment at its former state; if AC1 contains nonzero, it forces rescheduling.

If the message you have sent to the task could affect the environment, you may want to force rescheduling on exit from the interrupt routine.

If the task for which a .IXMT message is intended has issued a .REC for the message, RDOS immediately readies that task, even though the task environment is frozen. Contents of all accumulators are destroyed upon return from .IXMT, so your program must restore AC3 and AC2 (if unmapped), before it tries to exit from the service routine via .UIEX. (See Chapter 7, *Servicing User Interrupts*, and .UIEX.)

Required input

AC0 - Location of the message. (The contents of this location must be zero before you invoke .IXMT.)

AC1 - The nonzero message you want to transmit.

Format

```
.IXMT
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
43	ERXMT	Message address is already in use.
115	ERXMZ	Zero message word.

Receive a Message (.REC)

This command returns a message in AC1 that another task (or interrupt service routine) has posted by a transmit command, and restores the contents of the message address to all zeroes. The message address must be lower than 2^{15} (bit 0 must not be set).

If a task issues .REC and no other task has posted a message to the message address, the receiving task remains suspended until the message is sent. If the message has already been issued and if the receiving task has not also been suspended by .ASUSP (or .TIDS), control returns to the Task Scheduler. Otherwise the task remains suspended until you ready it with .ARDY. If several tasks attempt to receive the same message, only the highest priority task will receive the message.

Required input

AC0 - The message address.

Format

```
.REC
normal return
```

There is no error return from a .REC command; RDOS returns AC2 unchanged.

Locking a Process Via the .XMT/.REC Mechanism

You can use .REC and .XMT to lock and unlock a process or database which several tasks share, and to prevent more than one task at a time from accessing the database or the process path. To do this, your program must define a synchronization word, the message location, to which all tasks will issue a .REC. The task in control of the locked resource then issues .XMT to the synchronization word when it wants to open the resource to the other waiting tasks. RDOS then readies the highest priority task waiting to receive (.REC) the synchronization word and gives it unique control of the resource. This task, in turn, can use the resource until it unlocks the resource and so on.

Your program must initialize the locking facility before any tasks can use it. It can do this either by setting the synchronization word initially to a nonzero value, or by having an initialization task issue .XMT to the synchronization word.

User Overlay Management

In a multitask environment, different tasks can compete for an overlay node, or they can use the same overlay simultaneously. To handle this properly, you must consider several things that were unnecessary in a single-task environment.

*IXMT and certain other user interrupt calls are not really task calls, since you can issue them only from an interrupt-processing routine. When you use them, the task scheduler and task environment are in suspension. See Chapter 7.

You should use .TOVLD, the task call version of .OVLOD, to load overlays in a multitask program. If you use .OVLOD, only one task in the program can load overlays; moreover, you can't use .OVLOD and .TOVLD in the same program. If you choose .TOVLD, the maximum number of overlay nodes you can reserve is 125.

As part of its resource management activities, the Task Scheduler maintains a record called the overlay use count (OUC) of the number of tasks using a currently-resident overlay. It keeps the OUC in an overlay directory which the loader creates for each node in your program. (See Appendix E).

A ready task can request an overlay (via .TOVLD) either by segment and overlay number, or by symbolic name, if you assigned the name via a .ENTO pseudo-op (see Chapter 4, *User Overlays).* Whenever a task requests an overlay, RDOS checks the overlay directory and the overlay request for certain parameters. If the parameters permit, it loads the overlay into the node, increments the OUC by 1, and gives control to the Scheduler. If the parameters don't allow the load, RDOS suspends the calling task (bit T of TPRST) and control goes to the Scheduler; the task will be readied and the overlay loaded when the parameters permit. All this happens each time a task requests an overlay load.

Every time a task releases a resident overlay (via .OVREL, .OVEX, or .OVKIL), the overlay's OUC is decremented by 1. The overlay which currently occupies the node is not released (allowing a task to load another overlay into the node) until the OUC reaches 0. When it reaches 0, another task can load a new overlay; ~~this~~ will set the OUC to 1.

An unconditional disk (not virtual) overlay request guarantees a fresh copy of the overlay. A conditional overlay request loads the overlay only if it is not already in memory; if the overlay is resident, RDOS increments the OUC by 1. Conditional loads can save time, but you may use them only for reentrant overlays. As mentioned in Chapter 4, we recommend that all your overlays be reentrant; if any overlay is not, a task which wants it must load it unconditionally.

Load a User Overlay (.TOVLD)

This command requests the use of the appropriate overlay node and the loading of the overlay whose node/number you specify in AC0.

If you didn't give a symbolic name to the overlay via .ENTO before loading the program, you must pass the node number which it will occupy in the left byte, and its overlay number in the right byte. The node number corresponds to the segment number within the overlay file. The first segment, number 0, was defined by the first set of brackets in the RLDR command line; it corresponds to node 0 in memory.

The overlay number is the relative position of the overlay within its segment. Segment 0's overlays are numbered 0, 1, ...n. The second segment loaded is segment 1, corresponding to node 1; its overlays are 0,1, ...n, and so on.

You can specify either a conditional or unconditional load (described above) in AC1. If the load request is conditional and the node is free, RDOS loads the overlay. If the node already contains the requested overlay, RDOS return to the Scheduler immediately. Because another task is also using the overlay, it must be reentrant. If another overlay is currently in the node, with a nonzero OUC, the caller is suspended until the node becomes free.

If the load request is unconditional and the node is free, RDOS loads the overlay whether it is currently memory resident or not. If the overlay use count has not gone to zero (freeing the node), the caller is suspended (bit T of TPRST) until the node becomes free.

Figure 5-3 shows the sequence which the system follows when you issue .TOVLD.

Required input

AC0 - Overlay node/number word.

AC1 - For a conditional load, pass 0. For a unconditional load, pass -1.

AC2 - The channel number on which you opened the overlay file (see .OVOPN, Chapter 4).

Format

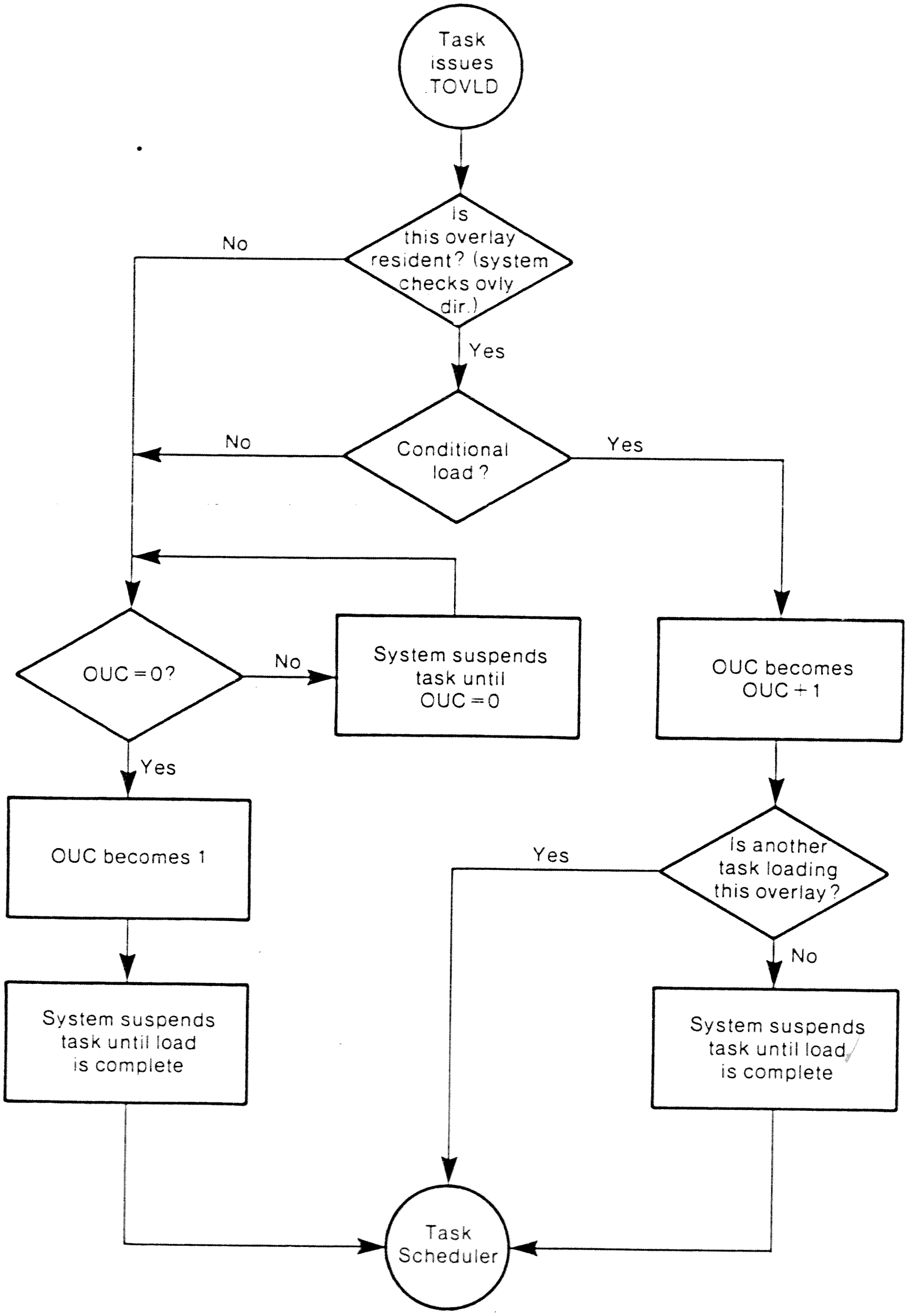
```
.TOVLD
error return
normal return
```

Note that you must pair all overlay load requests with an eventual overlay release (.OVREL/.OVEX/.OVKIL) or the node will be reserved indefinitely.

Under certain conditions (such as a nonmatching save and overlay file), the left byte of AC2 may be nonzero on an error return.

Possible errors

AC2	Mnemonic	Meaning
37	EROVN	Invalid (nonexistent) overlay name or segment.
40	EROVA	Overlay file is not a contiguous file.
101	ERDTO	Ten-second disk timeout occurred.


Task issues .TOVLD
Is this overlay resident? (system checks ovly dir.)
No
Yes
Conditional load ?
No
Yes
OUC=0?
No
System suspends task until OUC=0
OUC becomes OUC+1
Yes
OUC becomes 1
Is another task loading this overlay?
Yes
No
System suspends task until load is complete
System suspends task until load is complete
Task Scheduler

SD-00540

Figure 5-3. .TOVLD Logic Sequence

Release an Overlay (.OVREL)

This command decrements the overlay use count and releases the node if the use count equals zero. The overlay which you wish to release must not issue this command.

Required input

AC0 - Overlay node/number word.
Left byte: node number
Right byte: overlay number

Format

```
.OVREL
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
37	EROVN	Invalid overlay node/number; or the overlay node is not occupied by the specified overlay.

Release an Overlay and Return to the Caller (.OVEX)

This command decrements the overlay use count and releases the node if the overlay use count equals 0. Additionally, control returns to an address specified by the caller - typically the return address of the caller if returning from a subroutine within an overlay.

Required input

AC0 - Overlay node/number word.

AC2 - Return address upon successful execution of this call.

Format

```
.OVEX
error return
```

Possible error

AC2	Mnemonic	Meaning
37	EROVN	Invalid overlay number; the overlay node is not occupied by the specified overlay.

Kill the Calling Task and Release its Overlay (.OVKIL)

.OVKIL kills the caller and decrements the overlay use count; it also releases the node if the OUC equals 0. This is the normal method of terminating a queued, overlayed task. The overlay which you wish to release can issue this call.

Required input

AC0 - Overlay node number in the left byte, overlay number in the right byte.

Format

```
.OVKIL
error return
```

Possible error

AC2	Mnemonic	Meaning
37	EROVN	Invalid overlay number.

Enqueuing Tasks

Queue a Memory-resident or Overlay Task (.QTSK)

This command periodically initiates a task and queues it for execution. If the task resides within an overlay, this call loads the overlay. You need not issue .TOVLD for an overlayed task, but the .QTSK mechanism requires that you declare .TOVLD external (.EXTN) in the program. If there is no TCB currently available for the creation of the new task, RDOS will carry out this call as soon as a TCB becomes available. If two tasks are queued for execution at the same time of day, the higher priority task will receive control first. (The EXAMPLE program in Appendix D shows .QTSK and overlays in action.)

A task created and queued by .QTSK resembles any other task, and it is your responsibility to kill it or suspend it after it has done what you want. If it resides in an overlay, it can kill itself and release the overlay node via .OVKIL (if it doesn't release its node, no other task will be able to use the node).

If RDOS does not take the error return, control returns to the task issuing the call at the normal return based on the task's priority; the calling task is not suspended. When the queued task gets control, AC2 will contain a pointer to the Task Queue Table.

If your program doesn't declare either .TOVLD, .OVKIL, OVREL, or .OVEX as external (.EXTN), RDOS will execute a .SYSTM .ERTN, with AC2 equal to ERQOV.

.QTSK doesn't require input to AC0 or AC1, but it does require you to build a table of specifications for the new task, and to input the starting address of this table in AC2. The table must be QTLN words long (these symbols are defined in file PARU.SR, Appendix B), and have the following entries:

User Task Queue Table

Displacement	Mnemonic	Meaning
0	QPC	Starting address of task.
1	QNUM	Number of times to queue the task (-1 if the task is to be queued an unlimited number of times).
2	QTOV	Symbolic name or node number/overlay number (-1 for a memory-resident task).
3	QSH	Starting hour (-1 if the task is to be queued immediately).
4	QSMS	Starting second in hour (reserved but unused if QSH=-1).
5	QPRI	Task ID/task priority.
6	QRR	Rerun time increment in seconds.
7	QTLNK	System word.
10	QOCH	Overlay channel (unused by memory-resident tasks).
11	QCOND	Conditional/unconditiona load flag (unused by core-resident tasks).

For an example of .QTSK, see the application below.

Entry QPC must contain the entry address in the overlay or memory-resident task where control will be directed when RDOS raises the task to the executing state. QNUM is an integer value describing the number of times the task will be queued. The task will be queued QNUM times (or without limit if QNUM = -1) unless you issue the task call .DQTSK. This call halts the queuing of the specified task. RDOS decrements QNUM each time it queues the task.

QTOV must contain the overlay's .ENTO name, or its number, in the left byte, and overlay number in the right byte for overlay tasks; for memory-resident tasks, set this word to -1.

If you didn't assign a symbolic name to the overlay with .ENTO, you must use the segment/node number and overlay number assigned by the loader. Make sure that the values of QTOV correspond to the values assigned at load time.

Entries QSH, QSMS, and QRR all affect the time that RDOS will create the task. QSH sets the hour to execute, and QSMS set the second within that hour that the task will be created. If QSH contains -1, RDOS will create the task immediately.

If QSH occurs before the current time of day, or if QSH is greater than 24 hours and less than 48 hours, RDOS queues the task for the next day. If QSH is equal to (24*d) + h, RDOS will queue the task in d days.

QRR sets the interval (in seconds) between the times the task will be queued.

QPRI contains the task ID (if any) in its left byte and the task priority in its right byte. If a task with the same ID exists at the time that RDOS activates the task, the system will clear this task's ID number to zero. The system maintains QTLNK.

QOCH must contain the number of the channel on which you opened the overlay file with a previous .OVOPN call. QCOND must contain a minus one if you want the overlay load to be unconditional. QOCH and QCOND are unused by memory-resident queued tasks.

QAC2 is used as a temporary storage area by RDOS.

Required input

AC2 - Pointer to the task queue table.

Format

```
.QTSK
error return
normal return
```

On the normal return, AC2 will contain the contents of .QAC2.

Possible errors

AC2	Mnemonic	Meaning
50	ERQTS	Illegal information in Task Queue Table.
117	ERQOV	.TOVLD not loaded for an overlay queued task.

.QTSK Example

Let's look at a sample application of .QTSK in action: an airline arrivals/departures closed-circuit television display network, shown in Figure 5-4 (there is another example in Appendix D). One overlayed task checks a central control panel for each arrival and departure, and displays it, along with pending or recent arrivals and departures, on network screens throughout the terminal. The amount of traffic varies with the time of day, and .QTSK adjusts the interval at which the task checks the control panel. The following extracts from the main program show .QTSK code for 12:30 p.m., which is a comparatively slow time; .QTSK specifiès a 60-second check on the panel.

Dequeue a Memory-resident or Overlay Task (.DQTSK)

This call dequeues a task which task call .QTSK has queued for execution. In effect, the .DQTSK call bypasses the value which is currently stored in QNUM of the queued task's User Task Queue Table. If, at some later moment the task is requeued by a call to .QTSK, the queuing process will resume its normal course since .DQTSK does not actually modify the contents of QNUM.

Required input

AC1 - ID of the task to be queued.

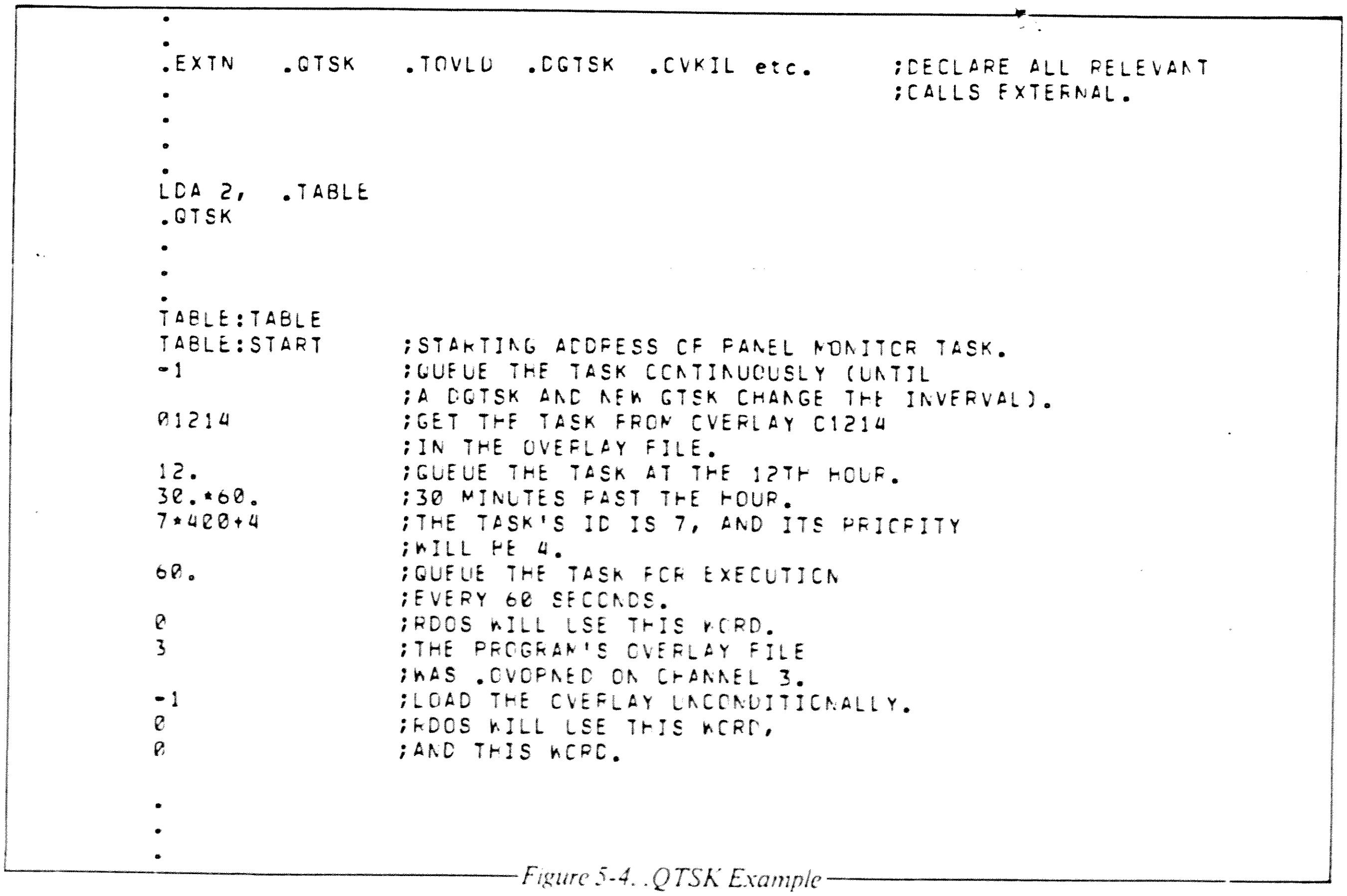
Format

```
.DQTSK
error return
normal return
```

Upon a normal return, AC2 returns the base address of the task's queue table.

Possible error

AC2	Mnemonic	Meaning
61	ERTID	Task ID error.



```
.
.
.EXTN    .QTSK    .TOVLD   .DQTSK   .OVKIL etc.      ;DECLARE ALL RELEVANT
.                                                   ;CALLS EXTERNAL.
.
.
.
LDA 2,   .TABLE
.QTSK
.
.
.
TABLE:TABLE
TABLE:START          ;STARTING ADDRESS OF PANEL MONITOR TASK.
-1                   ;QUEUE THE TASK CONTINUOUSLY (UNTIL
                     ;A DQTSK AND NEW QTSK CHANGE THE INVERVAL).
01214                ;GET THE TASK FROM OVERLAY O1214
                     ;IN THE OVERLAY FILE.
12.                  ;QUEUE THE TASK AT THE 12TH HOUR.
30.*60.              ;30 MINUTES PAST THE HOUR.
7*400+4              ;THE TASK'S ID IS 7, AND ITS PRIORITY
                     ;WILL BE 4.
60.                  ;QUEUE THE TASK FOR EXECUTION
                     ;EVERY 60 SECONDS.
0                    ;RDOS WILL USE THIS WORD.
3                    ;THE PROGRAM'S OVERLAY FILE
                     ;WAS .OVOPNED ON CHANNEL 3.
-1                   ;LOAD THE OVERLAY UNCONDITIONALLY.
0                    ;RDOS WILL USE THIS WORD,
0                    ;AND THIS WORD.

.
.
.
```

Figure 5-4. .QTSK Example

User/System Clock Commands

You can issue all system clock commands from either a single task or from a multitask environment. Since these commands are of little practical use in a single task environment, we present them in this chapter instead of in Chapter 3.

The following system calls permit your program to define, exit from, and remove a clock driven by the system Real Time Clock (RTC). This clock will suspend the environment at the intervals you specify, and pass control to the routine whose address you specify. You can exit from this routine and return to the environment via call .UCEX. You may not issue any system or task calls (other than .IXMT, .SMSK, and .UCEX) from this routine because RDOS freezes all multitask activity, just as it does for a user interrupt (see Chapter 7).

Any user clock routine executes in the interrupt world, not in program space, hence you should make sure that your routine is correct.

Delay Execution of the Calling Task (.DELAY)

This command suspends the calling task for the number of real time pulses indicated by AC1. You set the real time clock frequency at SYSGEN time (see .GHRZ, below).

The accuracy of .DELAY can be affected by three variables:

- The frequency of the Real-Time Clock, as set at SYSGEN;
- The priority of the issuing task, compared to other tasks;
- The priority of the issuing program (ground) compared to the other program.

RTC pulses are not synchronized with the .DELAY call; thus it may be unrealistic to request single-pulse delays. Single-pulse delay requests can be delayed anywhere between 0 and 1 RTC pulse.

Required input

AC1 - Number of RTC pulses.

Format

```
.SYSTM
.DELAY
error return
normal return
```

RDOS never takes the error return. You lose the contents of AC1 upon return.

Define a User Clock (.DUCLK)

This call defines a user clock, which will be entered at the intervals you specify in AC0. When this interval expires, RDOS suspends the Task Scheduler and multitask environment--if any-- and control goes to the address you specify in AC1. Each time control goes to this address, AC0 will contain a value indicating where control came from at the interrupt. AC0 will contain -1 if control came from the system while it was in an idle loop (i.e., awaiting an interrupt); it will contain 100000 if the other ground's program held control. If your program had control AC0 will contain the current PC.

When control passes to your user clock routine, AC3 will contain the address of the return upon entry to the user routine. In unmapped systems, you must use this address in the .UCEX command to return to the multitask environment.

Required input

AC0 - The integer number of system RTC cycles which you want to elapse between each clock interrupt.

AC1 - The address of your routine which will receive control when each interval expires. Note that you may not issue any system or task calls (except for .UCEX, .IXMT, or .SMSK) from this routine. Moreover, you must not issue assembly instruction INTEN in an unmapped system.

Format

```
.SYSTM
.DUCLK
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
45	ERIBS	A user clock already exists.
74	ERMPR	Address outside address space (mapped systems only).

Exit from a User Clock Routine (.UCEX)

When RDOS enters a user clock interrupt routine, it places the return address in AC3. In an unmapped system, RDOS requires this address to return to the multitask environment; therefore if your interrupt routine uses AC3, it must restore AC3 before issuing .UCEX.

(In mapped systems, RDOS ignores the value input in AC3 when you issue this call.) In all systems, RDOS will reschedule both the task environment and the program environment only if AC1 contains some nonzero value upon exit.

Control returns to the point where the .DUCLK interrupt occurred. You may issue this call in a single task environment.

Required input

AC1 - Zero to continue the environment; nonzero to reschedule.

AC3 - Return address to routine (unmapped systems only).

Format

```
.UCEX
```

Possible errors

none.

Remove a User Clock (.RUCLK)

This system command removes a previously defined user clock from the system.

Format

```
.SYSTM
.RUCLK
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
45	ERIBS	No user clock is defined.

Examine the System Real Time Clock (.GHRZ)

This system call returns a code for the Real Time Clock frequency in AC0.

AC0	Meaning
0	There is no Real Time Clock in the system.
1	Frequency is 10 HZ.
2	Frequency is 100 HZ.
3	Frequency is 1000 HZ.
4	Frequency is 60 HZ (line frequency).
5	Frequency is 50 HZ (line frequency).

Format

```
.SYSTM
.GHRZ
error return
normal return
```

Possible errors

none.

Managing Tasks by ID Number

Get a Task's Status (.IDST)

.IDST returns a code describing a task's status in AC0.

Required input

AC1 - The task's identification number.

Format

```
.IDST
normal return
```

The code returned in AC0 describes the task's status:

0	Ready;
1	Suspended by a .SYSTM call or .TRDOP;
2	Suspended by a .SUSP, .ASUSP, or TIDS;
3	Suspended by a .XMTW or .REC;
4	Waiting for an overlay node;
5	Doubly suspended by .ASUSP, .SUSP, or .TIDS and by .SYSTM;
6	Doubly suspended by .XMTW or .REC and .SUSP, .ASUSP, or .TIDS;
7	Waiting for an overlay node and suspended by .ASUSP, .SUSP, or .TIDS;
10	No task exists with this ID number.

RDOS will return the base address (displacement TPC) of the task's TCB in AC2.

Possible errors

none.

Change a Task's Priority (.TIDP)

.TIDP changes the priority of the task whose ID you specify in AC1.

Required input

AC0 - The new priority (from 0 to 255 inclusive) in the right byte (bits 8-15).

AC1 - ID of task.

Format

```
.TIDP
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
61	ERTID	Task ID error.

Ready a Task by ID Number (.TIDR)

.TIDR readies only that task whose identification number you place in AC1. It resets bit U in word TPRST of this task's TCB, which was set by a previous call to .ASUP, .SUSP, or .TIDS. If the specified task's bit U of TPRST was already reset, RDOS takes the normal return.

Required input

AC1 - ID number of the task you wish to ready.

Format

```
.TIDR
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
61	ERTID	Task ID error

Suspend a Task by ID Number (.TIDS)

.TIDS suspends only that task whose identification number is input in AC1. It sets bit U in word TPRST of the specified task's TCB. If the task's bit U in word TPRST is already set, RDOS takes the normal return.

Required input

AC1 - ID number of the task you wish to suspend.

Format

```
.TIDS
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
61	ERTID	Task ID error (no task exists with the specified ID number).

Kill a Task by ID Number (.TIDK)

.TIDK kills only that task whose identification number is specified in AC1. RDOS will raise the task to the highest priority (0), place it at the end of that priority chain, and transfer it to a kill processing address (if any) or terminate it. If the task is executing a system call, the kill will not occur until the call is completed.

Required input

AC1 - ID number of the task you wish to kill.

Format

```
.TIDK
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
61	ERTID	Task ID error.

Task/Operator Communications Calls

This section describes two calls, .TWROP and .TRDOP, which a task can issue to communicate with the system console, STTO/STTI. You can use these calls to interact directly with tasks in your program via OPCOM commands (see next section), or you can use just the calls or OPCOM alone. To use either (or both) features, you must have selected operator messages during RDOS system generation. If your program will use operator message calls or OPCOM commands, you must specify an extra task in the load command line to provide a TCB for system use. The format of console commands is similar for the task calls and OPCOM messages.

Note that your program can't use both system and task versions of the operator message calls. The system versions, described in Chapter 6, are .WROPR and .RDOPR; the task versions, described below, are .TWROP and .TRDOP.

Write a Task Message to the Console (.TWROP)

.TWROP instructs the calling task to write an ASCII string to the system console, $TTO. The message can include up to 129 characters, including the required carriage return, form feed, or null terminator. RDOS always displays 2 exclamation points (!!), and a "B" or "F" before it displays the text string. The "B" or "F" indicates that a background or foreground task, respectively, issued the message. Depending on your input to .TWROP, RDOS then displays the task's ID number and the message. Thus the format of task messages to the console is:

!! F *[TID]* message or !! B *[TID]* message

If AC1 contains -1 when the task issues .TWROP, RDOS will display the three-character prefix (!!F or !!B) followed by the message which can be a string of up to 129 characters, including a required carriage return, form feed, or null terminator. If AC1 contains a value other than -1 on this call, the first four characters of the message area will be overwritten by the three octal digits of the task ID and one space. Text written to the console is the three-character prefix (as above), then the task ID, then the remainder of the message - a string of up to 124 characters, including the terminator.

More than one task can have an outstanding request to write task messages to the console. However, if you use task calls .TWROP/.TRDOP to write or read messages to or from the console, you cannot also use system calls .WROP/.RDOP within the save file. Several tasks can use the same message string (same byte pointer), but only if you suppress TID information.

.TWROP requires an extra TCB in the program.

Required input

AC0 - Byte pointer to area which holds the message. (If AC1 does not equal -1, this area must include a 4-byte null prefix to receive the task ID and space separator).

AC1 - -1 to suppress the task ID, other value to display ID (see above).

Format

```
.TWROP
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
74	ERMPR	Address outside address space.
120	EROPM	Operator messages not specified at SYSGEN time.

Read a Task Message from the Console (.TRDOP)

This task call prepares the calling task to receive a message from the system console, $TTI. The task issuing this call may reside in either the foreground or the background program areas, and more than one task message. a program may issue an outstanding request for a task message. However, if you use task calls .TWROP/.TRDOP to write or read messages to or from the console, you cannot also use system calls .WROP/.RDOP within the save file.

You must type CTRL E as the first character (echoed on the console as an exclamation point: !). If the cursor is not at column 0, type RETURN first. The second character must be either an F or B to indicate whether the task resides in the foreground or in the background. If you type some character other than F or B in column 2, RDOS sounds the console bell as a warning and accepts no further characters until you type an F or B.

After the F or B, type the ID of the task to receive the message, followed by a comma delimiter; then type the message itself immediately after the comma. The last character in the message string must be return, form feed, or null. The total including number of characters, including the CTRL-E, B or F, TID, comma, message, and terminator cannot exceed 132. The required format for an input message is as follows (angle brackets indicate an ASCII character):

<CTRL E> $\begin{Bmatrix} B \\ F \end{Bmatrix}$ TID, message)

If, after pressing CTRL E, you want to cancel the message transmission, depress RUBOUT. Pressing the RUBOUT key in any character position erases a command or message character, starting with the most recent character. On TTYs, a left arrow (←) is echoed for each RUBOUT.

Remember that you must have specified an extra TCB for .TRDOP in the load command line (two extra for both reads and writes). There must also be one TCB available for use by the system. RDOS will use this TCB to create a task to monitor the STTI keyboard for task-keyboard messages, allowing one or more tasks to issue .TRDOP.

RDOS can display two messages to indicate errors in messages intended for tasks. These messages and their meanings are as follows:

TID NOT FOUND	No task with the specified ID number was waiting for a console message.
INPUT ERROR	Nonnumeric character in task ID.

Required input

AC0 - Byte pointer to message area. RDOS will not transmit the task ID and comma to the message area.

Format

```
.TRDOP
error return
normal return
```

On the normal return RDOS gives the byte count in AC1 (including the carriage return terminator but excluding the task ID and delimiter).

Possible errors

AC2	Mnemonic	Meaning
42	ERNOT	Out of TCBs (i.e., there is no TCB available to monitor the console).
74	ERMPR	Address outside address space (mapped systems only).
120	EROPM	Operator messages not specified at SYSGEN time.

Task-Operator Communications Module (OPCOM)

The task-operator communications package, OPCOM, allows you to use console commands to check or change the status of tasks, and to run these tasks or queue them for periodic execution.

OPCOM is unrelated to the Command Line Interpreter (CLI), and has its own syntax and command definitions. OPCOM has a limited command repertoire since it -- unlike the CLI -- is part of the save file with which it is being used.

We have arranged the OPCOM commands alphabetically:

DEQ	Dequeue a queued task.
KIL	Kill a task.
PRI	Change a task's priority.
QUE	Queue a task for periodic execution.
RDY	Ready a task.
SUS	Suspend a task.
TST	Display a task's status.

OPCOM requires two modules: OPCOM and either OPMSG (unmapped) or MOPMS (mapped). RLDR will load these if you declare .IOPC external (.EXTN) in the program. You must also specify an extra TCB (or two for reads/writes) for RDOS use (this is not necessary if you included an extra TCB (or two) for the operator task calls). Also, you must have selected operator messages during SYSGEN.

The OPCOM module requires about 457_8 NREL words, and OPMSG (or MOPMS) requires about 472_8 NREL words; thus, you'll need a total of about 1150_8 words for any system.

Each OPCOM command evokes a task call which performs the desired function; therefore, you can find details on the internal operation of each command under its related call (e.g., QUE and .QTSK). Each OPCOM command requires that you enter a program number for the task; you specify this number in a table which you build for each task before initializing OPCOM. The program number can be the task ID number, or not. Certain commands require ID numbers, and others (RUN and QUE) require program numbers; to avoid confusion, we recommend that you use the task's ID number as its program number.

After initializing OPCOM, you can enter commands in the format specified under OPCOM command syntax; OPCOM will respond with the message "OK" if it has executed the command, or with one of four descriptive error messages.

Initializing the Operator Communications Package (.IOPC)

You must initialize OPCOM before you can issue any OPCOM commands.

If you are not going to issue OPCOM commands RUN and QUE/DEQ, AC0, AC1, and AC2 must each contain 0 when you make the .IOPC call. If you intend to use OPCOM commands RUN or QUE/DEQ, however, you must input three parameters to .IOPC.

The first of these parameters, passed in AC0, is the address of the queue area reserved for this call. OPCOM needs one queue area frame for each RUN or QUE command awaiting execution (the QUE command awaits execution until the task has been queued for the last time). The total queue area is *n* *QTLN words long where *n* equals the number of queue frames and QTLN is the queue frame size. QTLN is defined in PARU.SR. The queue area is managed exclusively by OPCOM.

You pass the second parameter in AC1. The left byte must contain the channel number on which you .OVOPNed the overlay file; if no overlay is involved, this byte must contain 0. The right byte of AC1 must describe the maximum number of different tasks which you will queue or run simultaneously; this corresponds to the value n in the discussion of queue areas. OPCOM can load overlay tasks on request, but your program must release each node used for these tasks by issuing .OVKIL, or .OVEX.

The last parameter, passed in AC2, is the base address (displacement 0) of the task table. This table consists of a series of five-word frames which describe each task to be RUN or QUEued. Build this table as follows:

Displacement	**Contents**
0	Program number.
1	Overlay symbolic name, or node (left byte)/number (right byte) (-1) if a core-resident task).
2	-1 only if unconditional loading is required
3	Task ID (left byte); task priority (right byte).
4	Task starting address.

The program number is distinct from the task ID, but you may assign the same value to them if you wish. You can modify the task priority by an appropriate OPCOM command. Terminate the task table *series* with a word containing -1.

Required input

To summarize, if you are not going to issue OPCOM commands RUN or QUE you must clear AC0, AC1 and AC2 to zero when you call .IOPC. If you want to issue RUN or QUE/DEQ commands, you must pass the following parameters to .IOPC:

AC0 - Queue area address.

AC1 - Left byte: overlay channel (or zero). Right byte: maximum number of queues.

AC2 - Task table address.

Format

```
.IOPC
error return
normal return
```

Possible errors

AC2	**Mnemonic**	**Meaning**
42	ERNOT	Out of TCBs.
120	EROPM	Operator messages not specified at SYSGEN time.

OPCOM Command Syntax

OPCOM has been designed to accept a limited number of keyboard commands to keep the command processor small (it must always remain a resident part of the save file). All OPCOM commands have the following fixed format:

<CTRL E> $\begin{Bmatrix} B \\ F \end{Bmatrix}$ *, command, task, [arg_1, ...arg_n]

You enter <CTRL E> by pressing the CTRL and E keys simultaneously. If the cursor is not at column 0, type RETURN first. You must then type either "B" of "F" to indicate whether the save file being commanded is in the background or the foreground. Both background and foreground programs use STTI/STTO. Immediately following the "B" or "F", type an asterisk followed by a comma.

Type the OPCOM command immediately after the comma. Follow the command with a comma and one or more task arguments; separate multiple arguments by commas. Terminate the command line with a carriage return. Note that the command structure is rigid; if you depart from the command format (e.g., use spaces or delimiters), OPCOM will reject the command and display the error message

INPUT ERROR

on the console.

When OPCOM has executed a command, it prints the message

!! $\left\{\begin{matrix}B\\F\end{matrix}\right\}$ OK

on the console.

Dequeue a Previously-Queued Task (DEQ)

The DEQ command dequeues the previously-queued task, whose ID you specify as an argument.

The *task ID* argument must be an octal integer in the range 1-377; it cannot be 0. After executing the command, OPCOM displays the message "OK"; you can then issue another command. If OPCOM cannot execute the command, it will display one of two error messages and await another command.

Format

<CTRL E> $\left\{\begin{matrix}B\\F\end{matrix}\right\}$ *,DEQ, task ID)

Possible errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

Kill a Task (KIL)

This OPCOM command immediately kills the task whose ID you specify as an argument.

The *task ID* argument must be an octal integer in the range 1-377; it cannot be 0. After executing the command, OPCOM displays the message "OK"; you can then issue another command.

Format

<CTRL E> $\left\{\begin{matrix}B\\F\end{matrix}\right\}$ *,KIL, task ID)

Possible errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

Change a Task's Priority (PRI)

The PRI command changes the specified task's priority to the priority given as an argument. The *task ID* and the *new priority* arguments must each be an octal integer within the range 1-377. After executing the command, OPCOM displays the message "OK" on the system console; you can then issue another command.

Format

<CTRL E> $\left\{\begin{matrix}B\\F\end{matrix}\right\}$ *, PRI, task ID, new priority)

Possible errors

Message	Meaning
INPUT ERROR	New priority exceeded 377_8, or syntax error detected.
TID NOT ACTIVE	No task with the specified task identification number was found.

Queue a Task for Periodic Execution (QUE)

QUE creates and periodically executes a task for execution with task call .QTSK logic. The task may be either memory-resident or an overlay.

If the task resides in an overlay, the QUE command will load that overlay. If there is no TCB currently available for the creation of the new task, RDOS will carry out this command as soon as a TCB becomes available. If two or more tasks are queued for execution at the same time of day, the highest priority task will receive control first. After each time that this call creates and activates a new task, you must ensure that the system kills or suspends this task. If the task resides within an overlay, your program must release the node after the task has executed; if it does not, no other task will be able to use the node.

Upon successful completion of this command, OPCOM will display the message "OK" on the system console; and you can then issue another command. If OPCOM cannot execute the command, it displays one of four error messages and awaits another command.

Format

<CTRL E> $\begin{Bmatrix}B\\F\end{Bmatrix}$ *, QUE,program#,[*hour*], [*minute*]↑)

[*second*], [*repeats*], interval [*,priority*]

The entries within brackets are optional.

The program # argument is the number that you chose when you initialized OPCOM with .IOPC. This argument may be the same as the task ID, or not.

If *hour* is less than the current time of day, or is between 24 and 48, RDOS will queue the task for the next day. If *hour* equals (24*d) + h, RDOS will queue the task in d days. To queue for midnight, queue for *hour* 24. To queue the task immediately, omit the *hour*, *minute*, and *second* arguments (but keep their comma delimiters in the command line).

The *repeats* argument defines the number of times the task will be executed, and *interval* determines the number of seconds to elapse between each time RDOS queues the task. The *interval* may not exceed 65,535 seconds (about 18 hours). If you omit the *repeats* argument, the task will be queued an unlimited number of times. (Even if you omit this argument, you must include its comma delimiter.)

The *priority* argument indicates the priority of the task you want to queue; it is optional because you have already specified the task priority (along with other task information) in the task table input to .IOPC. If you give the priority argument, program # it overrides the priority you specified to .IOPC. The priority argument is an octal integer; all others are decimal.

Possible errors

Message	Meaning
INPUT ERROR	One or more required arguments are missing in the command string, or you specified an invalid priority argument.
PROG NOT FOUND	You did not issue the .IOPC call, or you did not define the program number in the .IOPC call (i.e., the program table is incomplete).
NO QUEUE AREA	You defined an insufficient number of queue area frames in the call to .IOPC, hence no free queue area is available.
ILLOGICAL QUEUE	You input illegal information in the argument string (RDOS detected this when it passed to .QTSK).

Ready a Task (RDY)

This command readies the task whose ID you specify as an argument.

The *task ID* argument must be an octal integer in the range 1-377. After executing this command, OPCOM displays the message "OK" on the system console; you can then issue another command.

Format

<CTRL E> $\begin{Bmatrix}B\\F\end{Bmatrix}$ *, RDY, task ID!&

Possible errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

Execute a Task (RUN)

This call initiates either a memory-resident task or one within an overlay, and queues this task for immediate execution. If the task resides within an overlay, this command will load that overlay. If there is no TCB currently available for the creation of the new task, this command will be carried out as soon as a TCB becomes available. After this call creates and activates the task, you must ensure that this task is killed or suspended. If the task resides within an overlay, you must release the overlay node.

After completing this command, OPCOM displays the message "OK" on the system console; you can then issue another command.

Format

```
<CTRL E> {B} *, RUN, program #, [priority] )
         {F}
```

The program # argument is the number that was assigned to this program when you issued the initialization call .IOPC. This argument may or may not be the same as the task ID, and you must express it as a decimal integer.

The *priority* is an optional argument which indicates the priority of the task you wish to queue; since you already specified the task (along with other information) in the task frame input to .IOPC, you need not give a new priority. If you give the *priority* argument, it overrides the priority specified to .IOPC. The priority must be an octal integer.

Possible errors

Message	Meaning
INPUT ERROR	You did not specify a program number argument in the command string, or you specifed an invalid priority argument.
PROG NOT FOUND	You did not issue the .IOPC call, or you did not define this program number in the .IOPC call (i.e., the program table is incomplete).
NO QUEUE AREA	You defined an insufficient number of queue area frames in the call to .IOPC; therefore no free queue area is available.

Suspend a Task (SUS)

This command suspends the task whose ID you specify as an argument. This *task ID* argument must be an octal integer in the range 1-377. After executing this command, OPCOM displays the message "OK" on the system console; you can then issue another command.

Format

```
<CTRL E> {B} *,SUS, task ID )
         {F}
```

Possible errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

Display a Task's Status (TST)

The TST command displays a specified task's status on the console. After executing the command, OPCOM displays the following status on the console:

STAT=s, PRI=ppp

The status of the task, *s*, is an octal integer in the range 0-7 where the integers have the following meanings:

s	Meaning
0	Ready.
1	Suspended by a .SYSTM call or TRDOP.
2	Suspended by .SUSP, .ASUSP, or TIDS (SUS).
3	Suspended by .XMTW or .REC.
4	Waiting for an overlay node.
5	Doubly suspended by .ASUP, .SUSP, or .TIDS (SUS) and by a .SYSTM call.
6	Doubly suspended by .XMTW or .REC and by .SUSP, .AUSP, or .TIDS (SUS).
7	Waiting for an overlay node and suspended by .ASUSP, .SUSP, or .TIDS (SUS).

OPCOM returns the priority of the task *ppp* as from one to three octal digits in the range 0-377.

Format

<CTRL E> $\begin{Bmatrix}B\\F\end{Bmatrix}$ *,TST, task ID)

The *task ID* argument must be an octal integer in the range 1-377.

Possible errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

Example:

A typical series of console commands and messages might appear as follows:

!B*,RUN,1) *!!B OK* !B*,RUN2) *!!B OK*	Run task with program number 1 in the background, and OPCOM verifies execution of the command. Similarly, program 2 is run and is verified.
!B*RUN,3) *!!B INPUT ERROR*	An attempt is made to run 3, but OPCOM detects a syntax error (missing comma).
!B*,TST,1)	Operator requests status of program 1.
!!B STAT=1, PRI=002 !B*,SUS,1) *!!B OK*	OPCOM responds with status 1, *ready,* and priority 2. Operator suspends 1 and OPCOM verifies execution of the command.
!B*,TST,1)	Operator gets status of 1 again;
!!B STAT=2,PRI=002	status is *suspended by SUS.*
!B*,KIL,1) *!!B OK*	Operator Kills program 1, and after system verifies this, operator tries to test its status.
!B*,TST,1)	
!!B TID NOT ACTIVE	OPCOM responds with error message.

Disabling and Enabling the Multitask Environment (.SINGL and .MULTI)

In a normal multitask environment, ready tasks compete for CPU control according to their relative priority. Although you can assign the highest priority (0) to one or more tasks, rescheduling occurs on each system interrupt, or when the executing task issues a system or task call -- thus, in a multitask environment, even the highest priority task may be suspended. Under some circumstances, you may want a task to retain CPU control continuously. To give a task such control, RDOS provides the task call .SINGL.

When a task issues .SINGL, it disables the multitask environment and retains CPU control despite system calls and most task calls it issues; although interrupts continue, the Scheduler will allow the task to retain control. However, user interrupt routines defined via .IDEF continue to execute as usual. The privileged task retains CPU control until it restores the multitask environment by issuing task call .MULTI. The multitask environment is also restored if the task suspends or kills itself.

Generally, a task should not disable the environment unless it must be absolutely autonomous; certainly it should not do so if it relies on other tasks. If you must deny other tasks access to a critical resource, like a database, use the .XMT/.REC mechanism.

Neither .SINGL nor .MULTI affect the other program, in a foreground/ background environment.

As with other task calls, you must declare .SINGL and .MULTI external (.EXTN) in a source program if you want to use them.

Disable the Multitask Environment (.SINGL)

This call disables the multitask environment, and gives the issuing task continuing CPU control, despite its priority or any system calls (and most task calls) it issues. This can be useful for operations outside of user state (see Appendix C). There is no required input to .SINGL; there is no error return.

Format

```
.SINGL
normal return
```

Restore the Multitask Environment (.MULTI)

This call enables normal Scheduler operations and the multitask environment after they have been disabled by call .SINGL. There is no required input, nor an error return from .MULTI.

Format

```
.MULTI
```

Disabling the Task Scheduler

Generally, the RDOS multitask calls permit you to manage a multitask program with complete satisfaction; the task scheduler always gives CPU control to the higher priority ready task. In some instances, however, you may want to suspend the task scheduler briefly. For example, you might suspend rescheduling to control race conditions between several tasks competing for a single resource. Disabling the scheduler --even briefly-- is a drastic step. Note that disabling rescheduling will not affect system activities such as interrupt service. Moreover, RDOS will reactivate the scheduling function as soon as the issuing task loses control of the CPU, even though you may not yet have reenabled rescheduling explicitly. For instance, all system calls, .SUSP, and .KILL reenable scheduling.

Disable Rescheduling (.DRSCH)

This task call prevents rescheduling in this program environment until either you reenable scheduling explicitly or the issuing task loses control of the CPU. Issue task call .DRSCH with caution, since it disrupts the ordinary management of the multitask environment; the task that issues this call will retain control even though other higher priority tasks are ready. This call has no effect when scheduling is disabled.

Format

```
.DRSCH
normal return
```

Possible errors

none.

Reenable Rescheduling (.ERSCH)

Normally, the task scheduler is enabled and manages the multitask environment within its program. If you have suspended task scheduling by a call to .DRSCH and you have not issued a system call, you can reactivate the scheduler by issuing task call .ERSCH. This call has no effect when scheduling is enabled.

Format

```
.ERSCH
normal return
```

Possible errors

none.

Task Call Summary

NOTE: You must declare all task names external (.EXTN pseudo-op).

Table 5-1. Task Command Summary

.ABORT	Terminate a task immediately.
.AKILL	Kill all tasks of a given priority.
.ARDY	Ready all tasks of a given priority.
.ASUSP	Suspend all tasks of a given priority.
.DQTSK	Dequeue a previously-queued task.
.DRSCH	Disable the rescheduling of the task environment.
.ERSCH	Re-enable the rescheduling of the task environment.
.IDST	Get a task's status.
.IOPC	Initialize the Operator Communications Package (OPCOM).
.IXMT	Transmit a message from a user interrupt
.KILAD	Define a kill-processing address.
.KILL	Kill the calling task.
.LEFD	Disable LEF mode (Chapter 10).
.LEFE	Enable LEF mode (Chapter 10).
.LEFS	Get the LEF mode status (Chapter 10).
.OVEX	Release an overlay and return to the caller.
.OVKIL	Kill an overlayed task and release the overlay.
.OVREL	Release an overlay node.
.PRI	Change the calling task's priority.
.QTSK	Queue a core-resident or overlay task.
.REC	Receive a message from a task.
.REMAP	Trigger the MMPU for a window remap (Chapter 4).
.SMSK	Modify the current interrupt mask (Chapter 7).
.SUSP	Suspend the calling task.
.TASK	Initiate a task.
.TIDK	Kill a task by ID number.
.TIDP	Change the priority of a task by ID number.
.TIDR	Ready a task by ID number.
.TIDS	Suspend a task by ID number.
.TOVLD	Load a user overlay in a multitask environment.
.TRDOP	Read an operator message.
.TWROP	Write an operator message.
.UCEX	Return from a user clock routine.
.UIEX	Return from a user interrupt routine (Chapter 7).
.UPEX	Return from a user power fail service routine.
.XMT	Transmit a message to another task.
.XMTW	Transmit a message to another task and wait for its receipt.

OPCOM Commands

DEQ	Dequeue a previously-queued task.
KIL	Kill a task.
PRI	Change a task's priority.
QUE	Queue a task for periodic execution.
RDY	Ready a task.
RUN	Execute a task.
SUS	Suspend a task.
TST	Get a task's status.

End of Chapter

Chapter 6
Foreground-Background Programming

Thus far in this manual, we have described tools for using RDOS effectively in one program. Chapter 3 explained the essential calls, Chapter 4 offered some tools for extending useful memory, and Chapter 5 described multitasking; each chapter built upon the features explained in preceding chapters, but all were presented in the context of a single program.

This chapter describes dual programming - which means running two distinct programs simultaneously, and letting RDOS apportion CPU time and disk I/O time between them.

Initially, when you bootstrap RDOS, only the background is running; the CLI, running in background memory, displays its R prompt. You can then execute a foreground program directly, via CLI command EXFG, or you can execute a background program, which, in turn, may execute another program in the foreground via system call .EXFG.

How you handle dual programming will depend largely on whether or not your system has a hardware map to separate the two programs. Dual programming is safer and easier if you have a mapped system, and you can also use extended address space as described in Chapter 4. If your system is unmapped, you must configure a program for foreground execution by specifying starting ZREL and NREL addresses in the RLDR command line; nonetheless, with a little care, you can execute a program in both an unmapped foreground and background.

This chapter contains the following major sections and system commands:

- Dual programming- mapped systems
- Dual programming- unmapped systems
- Foreground/background system calls:

.EXFG	Execute a program in the foreground.
.FGND	See if the foreground is running, and check the status of the current program.
.ICMN	Define a program communications area.
.WRCMN	Write a message to the other program.
.RDCMN	Read a message from the other program.
.WROPR	Write an operator message.
.RDOPR	Read an operator message.
.EXBG	Checkpoint a mapped background program.

Related calls in other chapters are:

.MEM	Check the current program's NMAX (Chapter 3).
.MEMI	Change the value of NMAX (Chapter 3).
.EXEC	Swap or chain a save file (Chapter 4).
.ERTN and .RTN	Return the next higher-level program (Chapter 4).
.WRPR	Write-protect a memory block - mapped systems only (Chapter 4).

Introduction

The two programs that run under RDOS are called a *foreground* and a *background* program. These programs exist independently of each other, and each one has its own task scheduler. These two programs can have equal priority, or you can give the foreground program a higher priority than the background program. In this case, control will go to the background only when no task is ready in the foreground. When you need to run a real-time program with critical response time, run it in the foreground. The foreground will then receive a higher priority than the background, which, you can use for programs not requiring fast response (e.g., assemblies, compilations, and the like).

Foreground and background programs can communicate via a Multiprocessor Communications Adapter line, or they can each define a common *communications area* via .ICMN and transmit messages to the other via .WRCMN and .RDCMN. System call .FGND enables the background program to determine whether or not a foreground program exists. The foreground program can terminate its own existence via .RTN from level 0 (or you can terminate it by typing CTRL F from the background console), and it can release all its former memory.

Foreground and background programs can access common disk files and common directories. If foreground and background tasks are using the same directory, either task may release that directory without affecting the other task's use of the directory. If one program, F for example, releases a directory which is in use by another program, B, F will receive the error return with error code EROPD as an indication that the directory is in use by B. Nonetheless, RDOS will release the directory from F.

The foreground and background cannot use the same reserved device file simultaneously, nor can they spool data simultaneously to the same output device. Only the first ground to open the reserved device request will be able to use that device. Similarly, foreground and background programs should not issue simultaneous read commands to a common input device, since RDOS has no way to separate elements in an input data stream and divert them to two different programs.

If you have a mapped system, you can use all mapped system and task calls (.STMAP, .DEBL, and .DDIS (Chapter 3)), the write-protect, virtual overlay, and window map calls (Chapter 4), .EXBG (this chapter), and .STMAP (Chapter 7), as well as the special mapped calls, .WREBL and .WRPR. RDOS will treat any special mapped calls which you issue in an unmapped environment as no-ops, and will give control to the call's normal return.

Dual Programming-Mapped Systems

Mapped systems provide an absolute hardware boundary between the foreground and background programs. Moreover, the map provides both the foreground and background programs with a complete page zero (including auto increment/decrement locations) and a complete NREL memory area. You can run two CLIs concurrently in a mapped environment, if two consoles are available.

In mapped systems, all programs may use locations 16_8 and above, up to the limits of available memory, since each program has its own page zero. The system initally allots all memory blocks to the background program. You can change this initial memory allocation via the CLI command SMEM; you can check the current memory allocations via the CLI command GMEM or system call .MEM. Each program can change its own NMAX value via system call .MEMI.

Whenever a map violation occurs in an instruction which is not a call (e.g., an infinite defer, illegal address, or illegal attempt to reference a system device), RDOS outputs the contents of the program counter and accumulators as follows:

```
TRAP PC AC0 AC1 AC2 AC3
```

PC gives either the location of the instruction that caused the trap, or -1 if RDOS can't report a meaningful address. For example, you would get a -1 if your program tried a seriously illogical operation, like exiting from a user interrupt routine (.UIEX) when no such routine had been defined.

Following the TRAP message, RDOS creates a break save file (named BREAK.SV), places it in the current directory, and displays the message "BREAK" on the console. Control then goes to the next higher-level program whose UST location USTBR is set to a valid address. (See Chapter 3, *Keyboard Interrupts.)*

If you pass an illegal address to a system call, RDOS returns error code 74, ERMPR.

Writing interrupt routines for special user devices is slightly easier in a mapped system. If you want a user device to use the data channel, however, you must identify the device via system call .STMAP (Chapter 7).

When your program issues a .MEMI command in mapped environments, RDOS sets NMAX at whatever value is required by the specified memory increment, up to HMA, the highest memory address available to your program. Nonetheless, the map always allocates memory in blocks of 2000_8 words. Thus, for example , if NMAX is set at 40000_8 and you request a memory increment of 500_8, NMAX will become 40500 even though a total of 42000 memory words are reserved for the program.

You can build foreground save and overlay files for either ground in a mapped system in the same way you'd build save and overlay files for a single program background, since RDOS reserves an entire ZREL and NREL memory for each ground.

Executing Dual Programs in a Mapped System

The RDOS system bootstrap operation brings the CLI into execution in the background. At this point there is no foreground program loaded, so all available memory is allocated to the background. Thus, before you can issue any foreground command on a mapped machine, you must allocate memory to the foreground with the SMEM command.

After you have built an executable foreground save file (with optional overlays) you can load and execute in the foreground area by entering the CLI command EXFG savefilename). (Any background program can also execute a program in the foreground by issuing system call .EXFG.)

You can EXFG (.EXFG) any executable program, including a system utility command, or the CLI itself, and access it via a second system console, $TTI1/$TTO1. (If you use .EXFG instead of EXFG -- a utility command -- you must set up the foreground command file, FCOM.CM, as desribed in an appendix of the CLI manual.)

To execute a single system utility program in the foreground, issue the following command from the background console:

```
EXFG system-utility-command-stream )
```

To assemble source file ABC in the foreground with a cross reference and listing to the line printer, you'd type the command:

```
EXFG MAC ABC $LPT/L )
```

To execute the CLI itself or any other save file in the foreground, use the form:

```
EXFG program-name )
```

Any program executing in the foreground, may push other program levels into execution via the system call .EXEC.

The foreground program can terminate by issuing as many .RTN (.ERTN) system calls as it needs to pop through level 0 (if the CLI is not active in the foreground). This occurs when a single system program, executed at level 0 in the foreground, terminates its operation. Alternatively, you can terminate a foreground program by typing CTRL F on the background console. You must use this second method to terminate a program which has a CTRL-A and/or CTRL-C handler. When you issue CTRL A (or CTRL C) via the foreground console (if any), the foreground program will terminate if RDOS finds no interrupt processing address in USTIT/USTBR of the foreground UST, and if no higher level program contains such a processing address in its UST. Each system utility automatically issues a .RTN when it terminates to return control to the background (or, if executing in the background, to return control to the CLI).

Whenever the foreground program terminates via system calls .RTN or .ERTN, RDOS displays the message "FG TERM" on the console. The same message appears if you terminate the foreground by a CTRL F interrupt.

Checkpointing a Background Program

Checkpointing allows a foreground program to interrupt the current background program, run a new program in the background, and then restore the old background program.

Some processing applications will work better if the foreground program can make use of the background's resources in this way. One example of such an application is a mapped dual program system which contains a data collection program in the foreground and one of several system utilities in the background. In such an application, the foreground might occasionally need to execute a data reduction program in the background. Checkpointing the data reduction program into execution from time to time would fulfill this need. You can checkpoint via the mapped RDOS system call .EXBG, described in this chapter.

Dual Programming - Unmapped Systems

Unmapped systems must use software boundaries to separate the foreground and background program areas. You must define those boundaries before execution, in the RLDR command line.

Each boundary is a starting address for execution; the local /F switch defines the starting NREL address, and the /Z switch defines the starting ZREL address for execution.

Locations 20_8 through 37_8 are reserved for use by the background.

Building Foreground Programs

When you plan to run foreground and background programs in an unmapped system, bear in mind that the memory requirements of each will be critical.

Aside from this, and any possible foreground/background system calls, writing the source code for a foreground program doesn't require special consideration.

Depending on your application, you may want a *background* program to change NMAX (.MEMI, Chapter 3) if it will execute a specific program in the foreground via .EXFG.

After you've written and assembled your source program, configure it for foreground operation by including the starting ZREL and NREL boundary addresses in the RLDR command line. (It's good practice to check the ZMAX and NMAX requirements of the programs which you may want to execute simultaneously in the background; you can do this with the program load map or the SEDIT (or OEDIT) utility. The boundary information must include both NREL and ZREL address information in local switches F and Z.

An example of such a command line is:

```
RLDR 13000/F 250/Z R0 R1 [OV0 OV1, OV2]
```

This command creates a save file named R0.SV (containing binary files R0 and R1), and an overlay file named R0.OL (containing two overlays). When you load the save file into memory, RDOS will load its ZREL portion into locations 250_8 and above, and its NREL portion into locations 13016_8 and above.

When you're building programs for an unmapped foreground, always remember that the programs will be separated by soft boundaries only; hardware does not protect the address space of the two programs. Thus, for example, you must ensure that no background program attempts to return to a higher level background program requiring more core storage. If such a return is performed (e.g., via .RTN) and the higher level background program requires space now occupied by the foreground program, system failure results.

This situation would occur after the following sequence of program loads:

1. The CLI resides in the background and there is no foreground program in execution.
2. A background program (BGD), smaller than the CLI, is executed via the CLI on level 1.
3. BGD issues the foreground load command .EXFG, loading a larger program whose starting address immediately follows BGD's NMAX.
4. BGD issues .RTN, attempting to return to the CLI. The CLI, however, requires memory storage which the foreground program now occupies. System failure occurs.

You can avoid this by planning your program flow carefully.

Executing Dual Programs in an Unmapped System

The RDOS system bootstrap operation brings the CLI into execution in the background. At this point there is no foreground program, so all memory is allocated to the background. Once you have built an executable save file (see the previous discussion), you can execute it in the foreground.

You can execute a program in the foreground area via the CLI command EXFG or by the corresponding system call .EXFG. For either command to work, you must have loaded the foreground program with software boundary information.

To load and execute a program in the foreground, type the command:

```
EXFG program-name )
```

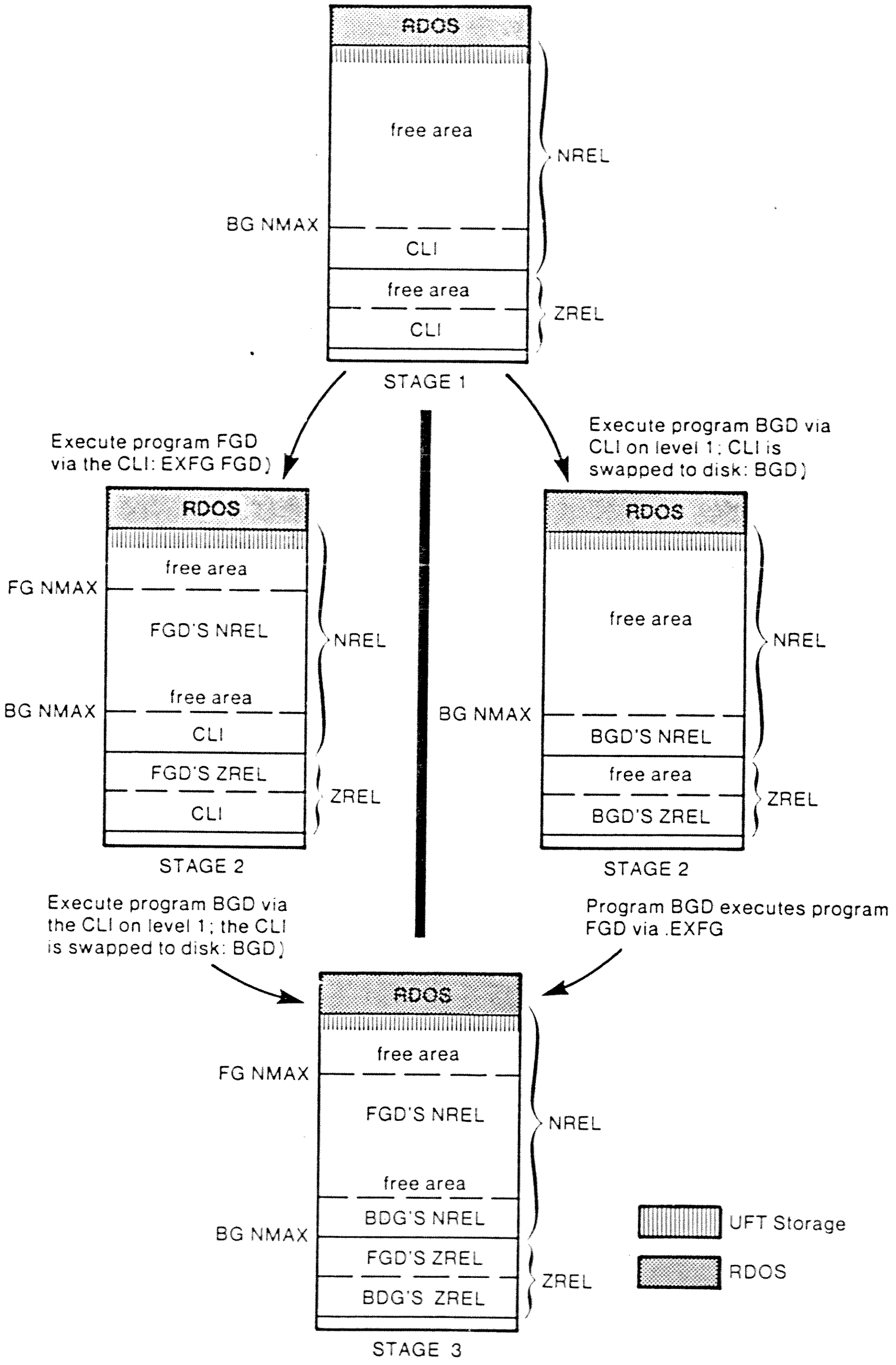
If the boundary requirements of the foreground program would overwrite any portion of either the CLI or the background program, RDOS won't load the foreground program.

If the foreground program's boundaries are valid, RDOS will load and execute it; the CLI will display its R prompt when execution begins. You can then try to execute a new background program via the CLI, swapping the CLI. If the program you wish to execute in the background requires more memory than is available, RDOS will not execute it.

You can terminate a foreground program by either typing the command CTRL F on the background console, or CTRL A (or CTRL C) on the foreground console, STTI1 (if any). This will terminate a foreground program as long as it has no interrupt processing address in USTIT (or USTBR) in its UST and no higher-level foreground program has such a processing address in its UST. The foreground program can release its memory to the background by issuing .RTN.

When you terminate the foreground via CTRL-F, (or CTRL-A/CTRL-C from TTI1), or when the foreground program yields its memory to the background via .ERTN (.RTN), the message "FG TERM" appears on the background console.

The following illustration (Figure 6-1) depicts two possible command sequences to produce foreground/background operation in an unmapped system. It uses two sample programs, FGN and BGD. The shaded areas in this illustration represent the storage areas occupied by User File Tables (UFT's). These are 45_8 -word data structures used by the operating system to record file and device information for each disk file opened on a channel. RDOS stores file information in a section of each UFT called a UFD; you can access UFD information with the .STAT system call discussed in Chapter 3. (In all mapped systems, UFTs reside in system space.)



To return to stage 1, FGD issues .RTN, relinquishing its memory to BGD. BGD then issues .RTN. From the console, typing CTRL-A, then CTRL-F would achieve the same end by interrupting the programs.

SD-00531

Figure 6-1. Loading Foreground and Background Programs in an Unmapped System

Execute a Program in the Foreground (.EXFG)

The .EXFG system command loads a program save file into foreground memory and transfers control to it. Only a background program can issue this command. In an unmapped system, you must have loaded the save file with boundary information (see preceding section). RDOS will pass the contents of AC2 to the foreground program.

Required input

AC0 - Byte pointer to the foreground program save file name.

AC1 - Appropriate starting address/foreground priority code.

Two possible addresses are allowed: The program starting address (USTSA), and the Debug III starting address (USTDA). The permissible codes input in AC1 are:

Code	Meaning to RDOS
0B15	USTSA; pass control to the highest priority ready task in the program. (Initially this is the program itself.)
1B15	USTDA; pass control to the debugger.
0B1	Give the foreground program a higher priority than the background.
1B1	Give the foreground and background the same priority.

Format

```
.SYSTM
.EXFG
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
4	ERSV1	File requires "Save" attribute.
12	ERDLE	File does not exist.
21	ERUFT	Not enough channels SYSGENed into the mapped system to satisfy the value specified in USTCH of the save file.
26	ERMEM	Attempt to allocate more memory than is available.
32	ERADR*	Illegal starting address.
53	ERDSN	Directory specifier unknown.
66	ERDNI	Directory not initialized.
70	ERFGE	Foreground already exists.
73	ERUSZ	Too few channels defined at load time or at SYSGEN time.
74	ERMPR	Address outside address space
101	ERDTO	Disk timeout occurred.

*RDOS will return ERADR if the code input in AC1 is illegal or if the required address is missing from the UST. This can occur if:

1. You didn't specify a starting address for the save file and you gave code 0B15.
2. You did not load the debugger as part of the save file and you gave code 1B15.

See if a Foreground Program is Running and Check Your Own Level (.FGND)

Use .FGND to determine whether or not a foreground program is running in the system, and at what program level the calling program is running.

Required input

none.

Format

```
.SYSTM
.FGND
error return
normal return
```

System call .FGND returns

AC0 - 1 if foreground found; 0 if no foreground.

AC1 - Code indicating the calling program's level, as follows:

Code	Meaning
1	Background level 0
2	Background level 1
3	Background level 2
4	Background level 3
5	Background level 4
6	Foreground level 0
7	Foreground level 1
10	Foreground level 2
11	Foreground level 3
12	Foreground level 4

Possible errors

none.

Define a Program Communications Area (.ICMN)

This call permits your program to define a contiguous area of up to 256_{10} words within its own address space to send or receive messages from another program. The foreground and background may each define one communications area.

Required input

AC0 - Starting address of the communications area.

AC1 - Size of the area in words.

Format

```
.SYSTM
.ICMN
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
62	ERCMS	Communications area exceeds the program size or would overwrite the system.
74	ERMPR	Address outside address space.

Write a Message to the Other Program (.WRCMN)

This call writes a message of up to 256_{10} words from the calling program (foreground or background) into the other program's communication area. The message sent may originate from anywhere within the sender program's address space.

Required input

AC0 - Word address of the start of the message.

AC1 - Word offset within the other program's communications area which will receive the message.

AC2 - Number of words to be sent.

Format

```
.SYSTM
.WRCMN
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
62	ERCMS	Message too large for communications area.
63	ERCUS	No communications area is defined in the other program.
74	ERMPR	Address outside address space.

Read a Message from the Other Program (.RDCMN)

This call lets the calling program read a message of up to 256_{10} words from another program's communications area. The receiving program may accept the message anywhere within its address space.

Required input

AC0 - Starting word address to receive the message.

AC1 - Word offset within the other program's communications area where the message originated.

AC2 - Number of words to be read.

Format

```
.SYSTM
.RDCMN
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
62	ERCMS	The size of the requested message exceeds the communications area size.
63	ERCUS	No communications area is defined in the other program.
74	ERMPR	Address outside address space.

Write an Operator Message (.WROPR)

This call instructs the calling program to write a text string to the system console, $TTO. There may be only one outstanding write-operator command in a program area. The message must consist of an ASCII string, less than or equal to 129 characters in length, including a carriage return, form feed, or null terminator. On the console, RDOS displays two exclamation points (!!), either an F or B, and then the message. The F and B indicates that the message came from the foreground or background program, respectively. Thus text strings output on the console are in one of two forms:

!!Ftext string or !!Btext string

You should not issue this call if you have also used OPCOM or the task message commands, .TWROP and .TRDOP, in the environment.

Required input

AC0 - Byte pointer to text string.

Format

```
.SYSTM
.WROPR
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
74	ERMPR	Address outside address space.
120	EROPM	Operator messages not specified at SYSGEN time.

Read an Operator Message (.RDOPR)

This call prepares the calling task to receive an operator message from the system console, $TTI; the task may exist in either the foreground or the background programs.

Before typing the message to the program, you must type CTRL E (echoed on the console as !); and an F or a B, to indicate whether a foreground or background program is to receive the message. RDOS will recognize CTRL E only if it is the first character in a line.

If no program has requested a console message, the TTY bell (if any) will ring when you press CTRL E; if the second character is anything other than an F or B (or rubout), the TTY bell will ring, and RDOS will accept no further input until you type an F or B.

If immediately after pressing CTRL E, you wish to cancel the message transmission, press RUBOUT instead of F or B. Pressing the RUBOUT key erases message characters, starting with the most recent character. RDOS echoes a left arrow (←) on teletypewriters; on CRT displays, it erases the last character each time you press RUBOUT. The last character in the message string must be a carriage return, form feed or null, and the total message length (including terminator) can be up to 132 characters.

Only one program (task) in each ground can have a read-operator message request outstanding at any one moment. You must not issue this call if you are using OPCOM or task operator message commands .TWROP and .TRDOP in this program environment.

Required input

AC0 - Byte pointer to message area.

Format

```
.SYSTM
.RDOPR
error return
normal return
```

On the normal return, RDOS returns the message byte count (including the terminator) in AC1.

Possible errors

AC2	Mnemonic	Meaning
74	ERMPR	Address outside address space.
120	EROPM	Operator messages not specified at SYSGEN time.

Checkpointing a Mapped Background Program (.EXBG)

Checkpointing is the practice of suspending one background program (the checkpointed program) temporarily so that you can execute a new background program.

Only a mapped foreground program may issue the checkpoint call. The foreground can also pass an optional one-word message to the new background program. There may be only one checkpointed program at a time; RDOS does not allow nested checkpoints.

Before you can checkpoint a background program, it must first be checkpointable; that is, it must not perform any multiplexor I/O, and it must not use any of the following system calls:

```
.DELAY
.RDOP
.IDEF / .IRMV
.DUCLK / .RUCLK
```

When a background program is checkpointed, RDOS displays the message

```
CP ENT
```

on the console. RDOS saves the following constants from the old background program, and restores them when it restores that program: priority, floating-point processor state, ongoing console input, and current directory.

During the checkpoint, the current directory for both grounds is the foreground's current directory. Thus if the new background program needs to access files, these must be in the current directory, or you must include directory specifiers to them.

You can give the new program one of two priorities: that of the foreground, or that of the checkpointed program.

Since RDOS preserves $TTI input to the checkpointed program, $TTI becomes unavailable for use by the new background program except via .RDOP. The new program can direct output to $TTO via system call .WROP, *write an operator message*.

The new program can restore the checkpointed program by issuing .ERTN or .RTN; you can also restore the checkpointed program by entering CTRL-A or CTRL-C from $TTI (if the new program's UST doesn't specify a different interrupt routine.)

On a keyboard interrupt, RDOS displays the message

CP INT

on $TTO. When the new program restores the checkpointed program normally (i.e., via .ERTN or .RTN), RDOS displays the message

CP RTN

on $TTO.

Required input

AC0 - Byte pointer to the new background save file name.

AC1 - 1B0: Give the new background program the same priority as the checkpointed program.

(Note that you should clear all other bits in AC1 to zero).

AC2 - (Optional) One-word message to the new background program.

Format

```
.SYSTM
.EXBG
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
2	ERICM	Attempt to checkpoint in an unmapped system.
4	ERSV1	File requires "S" (save) attribute.
12	ERDLE	File does not exist.
21	ERUFT	Not enough channels SYSGENed into the mapped system to satisfy the value specified in USTCH of the new background program.
25	ERCM3	Attempt to checkpoint a checkpointed background program.
26	ERMEM	Attempt to allocate more memory than is available.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
73	ERUSZ	Too few channels defined at load time or at SYSGEN time.
74	ERMPR	Address outside address space.
76	ERNTE	Program to be checkpointed is not checkpointable, or attempt to create two outstanding checkpoints.
101	ERDTO	Disk time-out occurred.

Example

In the following sequence, three sample programs complete a checkpoint procedure. The background program which we will suspend with a checkpoint is called BACK, the foreground program which will execute the checkpoint is called FORE, and the program we're checkpointing into the background is called COMP.

1. We execute FORE and BACK via CLI commands EXFG FORE) and BACK).

2. While both FORE and BACK are running, FORE issues .EXBG to COMP, checkpointing COMP into execution. BACK is suspended, but RDOS saves its current state, the FPU, all $TTI input to it, and remembers its current directory. The console displays the message CP ENT.

3. COMP reads data from some of FORE's files; it issues a few .WROP and .RDOP calls and receives replies from the console.

   Having done its work, COMP writes data to a file in FORE's current directory. It then tells FORE that it is done, via .WRCMN; FORE receives the message, reads COMP's data from the file, and continues.

4. COMP issues .ERTN, the console displays CP RTN, and BACK resumes execution from its original current directory; the console displays CP RTN.

End of Chapter

Chapter 7
User Interrupts and Power Failures

This chapter has two sections: The first describes how to establish and reference user interrupts; the second covers the system's handling of power failures. In some cases, you may want to write your own routine for handling power failures. If so, you can use calls from the first section.

The material in this chapter applies to both single and multitask environments, and - unless we note otherwise - to both mapped and unmapped machines.

This chapter includes the following calls:

USER INTERRUPTS

.IDEF Identify a user interrupt device. Exit from
.UIEX a user interrupt routine (this is a task call).
.IRMV Remove a user interrupt device. Change
.SMSK the current interrupt mask (task call). Set
.STMAP the Data Channel map (mapped only).

POWER FAILS

.UPEX Exit from a Power Fail Service Routine (task call).

Servicing User Interrupts

When the CPU detects an interrupt request, it suspends the current program and directs control to its device interrupt service program, INTD. (INTD is part of RDOS and is always memory-resident.) The CPU then directs control through the interrupt vector table to the proper device control table (DCT), using the device code as a guide. RDOS created DCT automatically for all devices specified at SYSGEN. If you want RDOS to recognize a non-SYSGENed device, you must write a routine for it, decide on a mask for it, and construct a DCT. An Application Note called *RDOS User Device Driver Implementation* describes user DCTs. Your three-word DCT provides an interface between the system and your service routine, by telling RDOS how to mask the device and where to find the service routine. It looks like this:

Entry Displacement	Mnemonic	Purpose
0	DCTBS	Reserved for system use.
1	DCTMS	Interrupt service mask.
2	DCTIS	Address of interrupt service routine.

DCTIS is a pointer to the routine which serves this specific device interrupt request. DCTMS is the interrupt mask that you want RDOS to OR with the current interrupt mask while the system is in your interrupt service routine. This mask establishes which devices -- if any-- will be able to interrupt the currently interrupting device. (The interrupts are on when you enter the routine but RDOS masks them for this priority device.) For more on interrupt masks, see the "Programmer's Reference Manual for Peripherals."

After a user interrupt occurs, control goes to your service routine; AC3 contains the return address required for exit from your routine, and AC2 contains the address of the DCT. The task call .UIEX exits from the routine and returns to the current environment. You can issue .UIEX in both single and multitask environments.

RDOS removes all user devices from the system when either a program swap or a chain occurs. When the system receives a user interrupt on a program level which has not identified the user device, it issues an NIOC to the device and then returns to normal program execution.

Whenever a device requiring special user service generates an interrupt request, the entire task environment halts until RDOS has serviced the interrupt. All tasks will resume former states when the environment restarts unless you transmit a message to one of them by means of the .IXMT call from the interrupt service routine. (See .IXMT, Chapter 5.) Rescheduling of the program and task environment can occur upon return from the routine, depending on the contents of AC1 in the return command (see .UIEX, below).

In addition to .IXMT, your user interrupt or user power fail routine can issue the task calls .SMSK, .UIEX, and .UPEX. You will find all of them below.

Identify a User Interrupt Device (.IDEF)

This call introduces to the system a device which you did not identify at SYSGEN time, but whose interrupts you want the system to recognize. (The .IDEF call places an entry in the interrupt vector table). A maximum of 10 user devices can be identified to the system at any moment. An .IDEF to any device also gives access to the CPU (code 77_8, so that you can do such things as disable and enable interrupts.

The number of free device codes (those which you can assign to user devices) depends on the hardware in your RDOS system. You can find system devices and their codes on the instruction reference card for your computer.

If your system has an IPB, and you want control when the watchdog timer times out, you must identify the timer via .IDEF (see Chapter 8). If you generated the current RDOS system without an IPB, and you .IDEF a device on device code 36, then RDOS will issue an NIOP to device code 37 whenever the real-time clock or power-fail monitor interrupts. (The IPB has device code 36, the watchdog timer device code 37). If you don't want this interaction, don't use device code 36 or 37 for a user device.

If you want to .IDEF a data channel device, your program must establish the data channel map for the device via .STMAP (mapped systems only).

Required input

AC0 - Device code of the new device.

AC1 - Address of the new device's DCT. (In a mapped system, this address must be in NREL space, i.e., above 400_8.) Mapped systems only: Set bit 0 to 1 if you want the new device to use the data channel.

AC2 - Number of 1K core blocks which the data channel map needs. This number must be one larger than the integer number of 1,024-word blocks used for data channel core buffers. (Applicable only to mapped systems where you set bit 0 of AC1 to 1 for this call.)

Format

```
.SYSTM
.IDEF
normal return
error return
```

Possible errors

AC2	Mnemonic	Meaning
36	ERDNM	Illegal device code (greater than 76_8). Device code 77_8 is reserved for for CPU which supervises the power monitor/auto restart option.
45	ERIBS	Interrupt device code in use or 10 user devices already identified.
65	ERDCH	Insufficient room in data channel map (unmapped only).
74	ERMPR	Address outside address space (mapped only).

Exit from a User Interrupt Routine (.UIEX)

This exit call returns control to a program environment after a user interrupt; you can use it only to terminate an interrupt service routine. In all systems, you can force rescheduling by passing a nonzero value in AC1; if AC1 contains 0 when you issue this call, the environment will resume without rescheduling. In a mapped system, RDOS ignores values input in the other ACs.

In an unmapped system, you must restore AC2 and AC3 to the addresses they had on entry to the routine. If you don't do this, the system will crash.

Required input

AC1 - Zero only to suppress rescheduling.

AC2 - Unmapped only - address upon entry to routine (DCT).

AC3 - Unmapped only - address upon entry to routine (return address).

Format

```
.UIEX
```

Possible errors

none.

Remove a nonSYSGENed Interrupt Device (.IRMV)

To prevent the system's recognition of an interrupt device which was identified by the .IDEF commmand, issue the .IRMV command.

Required input

AC0 - Device code for the device which you want to remove from the system.

Format

```
.SYSTM
.IRMV
error return
normal return
```

Possible error

AC2	Mnemonic	Meaning
36	ERDNM	Illegal device code (greater than 77_8, or attempt to remove a SYSGENed device.

Modify the Current Interrupt Mask (.SMSK)

Use this task call to change your interrupt mask for a service routine in both single and multitask environments. Whenever a user interrupt occurs, RDOS ORs the interrupt mask with the mask in the DCTMS of your DCT to produce the current interrupt mask. The .SMSK call allows your interrupt routine to change the old mask, and produce a new mask which is the logical OR of the old mask and a new value. .SMSK destroys the accumulators so you must restore them for the subsequent .UIEX.

Required input

AC1 - New value to be ORed with old mask.

Format

```
.SMSK
normal return
```

Possible errors

none.

Set the Data Channel Map (.STMAP)

Before a user device can employ the data channel in a mapped system, your program must issue .STMAP to set up the data channel map. This is a special map maintained by the mapping hardware for data channel use. .STMAP sets up the data channel map for the user device and returns in AC1 the logical address which you should send to the device.

This call is a no-op when issued in an unmapped system. In mapped systems, two possible error conditions may occur.

Required input

AC0 - Device code.

AC1 - Starting address (in your address space) of the device buffer.

Format

```
.SYSTM
.STMAP
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
36	ERDNM	Device code not previously identified via .IDEF as a data channel device.
74	ERMPR	Address outside address space.

Power Fail/Auto Restart Procedures

RDOS provides software support for the power fail/automatic restart option. When the system detects a power loss, it transfers control to a power fail routine which saves the status of all accumulators, the PC, and Carry.

If the console key is in the LOCK position when power returns, the system console will display this message when power returns:

```
POWER RESTORED
```

If possible, the system will then restore task state variables, resuming operating at the point of interruption. After this message appears, you may need to wait up to a minute for disks to come back on line.

If the console key is in the ON position when power returns, you must set all data switches to zero (down) and lift START when power returns. This outputs the POWER RESTORED message, restores task state variables, and resumes operation.

RDOS gives power-up restart service to the following system devices:

- teletypewriters and CRTs
- disks
- multiplexors
- line printers
- paper tape readers/punches
- card readers
- plotters

Character output devices may lose one or more characters during power up. Since power-up service for disks includes a complete reread or rewrite of the current disk block, you will use no disk information although you must wait for the unit's READY indicator to light. When power returns, RDOS restores the modem multiplexor lines when the user dials in. Line printers may lose up to a single line of information. Card readers may lose up to 80 columns of information on a single card. Devices requiring operator intervention (such as line printers, card readers, etc.) must receive such action if power was lost for an extended period of time.

RDOS does not provide power-up service for magnetic tape units or cassettes.

No power-up service is possible for semiconductor memory without a backup battery.

Power-Up Service for User Devices

If you want to provide service to user devices (identified by .IDEF), or if, for any reason, you want to be notified when a power fail occurs, write an interrupt service routine and issue an .IDEF call, passing 77_8 in AC0 and the address of your powerfail service-interrupt routine in AC1.

Use the .UPEX task call (below) to exit from your routine; this will force program rescheduling.

Exit from a Power Fail Service Routine (.UPEX)

Use task call .UPEX to return from a user power fail routine in single or multitask environments. Control returns to the location which was interrupted by a power failure. .UPEX has no normal or error return.

Required input

Mapped systems: none.

Unmapped systems:

AC3 - Return address which it contained on entry to the routine (when the system enters a user power fail service routine, AC3 contains the address required to return to your program.)

Format

```
.UPEX
```

Error Detection and Correction, ERCC (ECLIPSEs only)

If your system has the ERCC hardware option (available on ECLIPSEs only), it will correct all single-bit parity errors in memory. (Multiple-bit errors cause Exceptional Status, as described in Appendix F.)

RDOS detects all corrected memory faults from the ERCC. You can provide your own routine for handling memory errors by .IDEFing the error code ERCC. Use .UIEX to return from your routine, as follows:

Required input

AC3 - Return address upon entry to the routine (unmapped systems only).

Format

```
.UIEX
```

End of Chapter

Chapter 8
Multiple Processor Systems

This chapter describes managing a system which includes more than one Data General computer. There are two hardware options available to manage such a system: an InterProcessor Buffer (IPB), Model 4240, which allows two CPUs to communicate via full duplex lines, and a Multiprocessor Communications Adapter (MCA), Model 4206, which allows up to 15 CPUs to communicate via full duplex lines. The MCA also allows foreground and background programs to communicate at data channel speeds.

If you have an IPB or MCA, you can run your processors together, in a multiprocessor system; this system can use any or all of the features described in previous chapters of this book. If you have neither device, each CPU in your installation must run independently.

You can configure an RDOS system to support either (or both) an IPB or MCA during system generation by answering the SYSGEN questions about these devices correctly.

The IPB provides one full-duplex line for sequential and line I/O between two processors. It also provides a half-duplex line for RDOS; this enables RDOS to assure that systems sharing disk partitions do not simultaneously modify the SYS.DR or MAP.DR of any partition. The IPB also provides an interval timer which permits each processor to monitor the activity of the other. If either processor fails to service its real time clock periodically, the timer alerts the other processor.

Note that on a hardware level, each *shared* disk must be installed with the same device code and unit name. If one processor has a disk hard-wired as the first controller, DP0, the second processor must also have the disk hardwired as the first controller, DP0. Both processors will reference the disk as DP0. If a disk is *unshared*, it must have a unique device name.

Both sides must run with an RDOS of the same revision level for IPB support to work. If not, one side or the other will probably enter Exceptional Status.

IPB support maintains the integrity of system files and disk file structures, but does not provide protection for the *contents* of user files. Thus if both sides try to write to the same file at the same time (including, of course, read-modify-write), one file, or fractions of both may be lost.

A typical IPB system consists of two CPUs operating independently. This system permits each CPU to have a foreground and background program; programs in both CPUs can access files in the same disk partition. The IPB maintains the integrity of SYS.DR and MAP.DR in common disk partitions and allows each CPU to monitor the other's activity.

Another dual-processor application might use the IPB to back up a critical real-time program. In critical real-time situations, redundancy helps safeguard the total system, and allows it to continue running even if a CPU fails. One example of a fail-safe IPB application is a main system which runs the critical process, while a back-up system stands ready to assume the main system functions if the main system fails. While it is standing by, the back-up system runs lower-priority jobs such as data analysis, summary reporting, and program development. If the main system fails, the interval timer detects this failure and signals the back-up system to take control.

The MCA does not have an interval timer, nor does it allow CPUs to share disk directories, but it does enable up to 15 CPUs to communicate via their data channels. Each MCA controller supports up to 15 separate lines and each MCA line provides asynchronous full-duplex communications links for sequential I/O. Each line is a filename, which your program can access via system calls, and which you can access via CLI commands. You can also transmit an entire RDOS system via the special CLI command MCABOOT. Each MCA line offers high-speed interprogram or interprocessor communications with little processor overhead.

RDOS itself does not use the MCA. Unless you generate RDOS with IPB support, it will not maintain the integrity of a partition accessed by more than one processor.

To run a multiprocessor system under either IPB or MCA, each CPU must boot up an operating system in a separate disk partition, and each partition must have its own copy of an RDOS system and CLI files CLI.SV, CLI.OL, and CLI.ER.

Interprocessor Buffer (IPB) Programming

Interval Timer

The Interprocessor Buffer (IPB) hardware features an interval timer which tells one processor that the other processor has stopped. Specifically, the timer generates an interrupt request if either processor fails to service its real-time clock every second. RDOS treats this interrupt request as a user interrupt. You can write routines to identify the interrupt via system call .IDEF (Chapter 7). The device code of the interval timer is 37_8.

An interval timer interrupt indicates to RDOS that the other processor has stopped; hence you shouldn't use IDEB, or any other program that suspends interrupts for extended periods while both processors are running.

Dual Processor Program Communications

IPB hardware also allows the two processors to communicate via a full-duplex line. This communications link permits a user program running in either processor to read or write line or sequential I/O to the other processor, via special filenames.

The filenames for the read and write operations are:

SDPI - Input dual processor link (device code 40_8).

SDP0 - Output dual processor link (device code 41_8).

Each side has a SDPI and a SDPO; each side's SDPO is connected to the other side's SDPI. Thus one side's SDPO writes to the other side's SDPI.

So, if CPU0's program wanted to write to CPU1's program, program 0 would write to $DPO and program 1 would read from $DPI. Simultaneously, program 1 could write a message to program 0 via its $DPO. Each $DPO is a spoolable device. (Program 1 should issue the read request before program 0 issues the write, or a character will be lost.)

IPB Example

In this example, the main program (P) monitors and controls a real-time environment, and a secondary program (S) stands by to take over if P fails. A special restart task will bootstrap a system for S via system call .BOOT, described below.

P, the control program, runs in the foreground of one CPU, while less critical programs run in the backround (P could also run in the background of a single-ground environment).

As the primary program (P) monitors and controls the real-time environment, it sends periodic status reports to a log file on P's disk so that, in the event of its failure, S can seize control and maintain continuity.

The backup program, S, runs in the second CPU, in either a single- or dual-ground environment. At its very beginning, S creates a highest-priority restart task, which suspends itself by issuing a .REC to the user's interval timer interrupt routine. This interrupt routine issues .IXMT to wake up the restart task, which then bootstraps S.

If the main CPU, running P, develops a problem, the interval timer will generate an interrupt, and the interval timer service program in S will ready the restart task. The restart task will then close all files in program S, release all S's directories, reset I/O and bootstrap a new RDOS system, identical to P's. Having bootstrapped this new system, S reads P's status reports to determine where it stopped and proceeds to monitor and control the real time environment.

Bootstrap a New Operating System (.BOOT)

This call executes an orderly shutdown of the current RDOS system, and bootstraps the system you have indicated by a byte pointer in AC0.

Specifically, a .BOOT resembles the CLI command BOOT - it closes all background and foreground files, releases their directories, and resets all I/O; it then bootstraps the new system, which must exist in a secondary or primary partition. If the byte pointer specifies a link entry to the new system, you must also link the new system's overlay file and initialize all partitions involved in the resolution chain.

When you bootstrap a system conventionally, it asks questions about the date and time, and then invokes the CLI. If the data switches are all up or the switch register contains -1, then RDOS searches for a file named RESTART.SV; if it doesn't find RESTART.SV, it invokes the CLI. If a program issues .BOOT, and the data switches are up or register contains -1, *and* the new system can find RESTART.SV, then the new system will come up automatically, with the default date and time (January 1, 1968); then .BOOT will chain control on level 0 to RESTART.SV. If RESTART.SV doesn't exist, .BOOT will ask the conventional log-on questions.

Initially, RESTART.SV does not exist; you must create it to .BOOT a system without operator intervention. It could also be the name of the user program itself, or LINKed to the program name. If the current date and time are important to the real-time process, you must find some way to get them to the new program - perhaps via RESTART.SV itself, if the old program periodically stored date/time data in a file which RESTART will read when it gets control.

Required input

AC0 - Byte pointer to name of new operating system.

Format

```
.SYSTM
.BOOT
error return
```

There is no normal return, since upon the normal completion of this call BOOT will receive control, and pass control to the new operating system.

Possible errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	File name does not exits.
23	ERRTN	File RESTART.SV does not exist, yet data switches were set for restarting without operator intervention.
53	ERDSN	Unknown directory specifier.
74	ERMPR	Address outside address space (mapped systems only).
101	ERDTO	Disk timeout occurred.
107	ERSFA	Spool file is active.

Multiprocessor Communications Adapter (MCA) Programming

Data Transmissions

The type 4206 Multiprocessor Communications Adapter receiver/transmitter (MCAR/MCAT) allows programs to communicate over full duplex lines, in blocks of up to 8192 bytes, via the data channel. Each program can exist within the program space of a single CPU, or within up to 14 other CPUs, or both. A second 4206 receiver/transmitter, MCAR1/MCAT1, provides up to 15 similar additional communications links. Each CPU may communicate with any other CPU.

Depending on whether it is transmitting or receiving, each MCA line is a file name of the following form:

```
MCAT:rr                  (MCAT1:rr)
              or
MCAR:tt                  (MCAR1:tt)
```

where *rr* represents a receiver unit number from 1 through 15, and where *tt* represents a transmitter unit number in the range 0 through 15. Thus, four CPU's, each running foreground and background programs, could have ten possible line connections (see Figure 8-1).

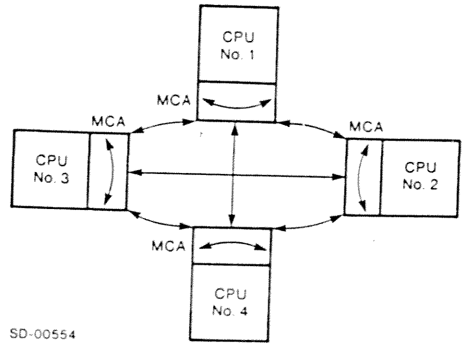


Figure 8-1. Multiple Processor Line Connections

If CPU 1 wanted to read (receive) from CPU 3, each unit would have to issue the following sets of instructions:

```
CPU 1
.OPEN n          ;OPEN MCAR:3
.RDS n           ;WAIT FOR THE DATA,
                 ;READ IT WHEN SENT.

CPU 3
.OPEN n          ;WRITE (TRANSMIT) TO
.WRS n           ;WRITE (TRANSMIT) TO
                 ;THE RECEIVER LINE.
```

CPUs 1 and 3 are operating under distinct RDOS systems. Thus, in the illustration, above there is no relationship between channel *n* for unit 1 and channel *n* for unit 3.

A receiver can request a transmission from *any* transmitter by issuing a read call to transmitter 0 (filename MCAR:0). After a receiver issues this read call, any transmitter can write to it. Thus if a program in CPU 1 had issued three receive requests, to MCAR:1, MCAR:3, and MCAR:0, it would receive transmissions from three sources: from its own machine (transmission from the other ground), from a program in CPU 3, and from any other program that wanted to transmit to it. Each transmitter would transmit by issuing a write to MCAT:1.

All messages must begin on a word boundry, and the receive and transmit byte counts must match. To transmit an end of file, you can transmit a zero-byte message (e.g., a .WRS of zero bytes).

A timeout can occur only in an MCA transmitter; a receiver can wait indefinitely. The timeout period ranges from about 200 milliseconds to about 655 seconds. The default timeout is 655 seconds, but you can select a shorter timeout period when you .OPEN the MCA line and issue a write sequential. See Chapter 3, .OPEN and .WRS, for details.

Get the Current CPU's MCA Number (.GMCA)

Your program can get the MCA unit number of its CPU by issuing system call .GMCA. It can then communicate this number to programs running in other CPUs.

Required input

AC0 - MCA transmitter octal device code (6 for MCAT, 46 for MCAT1).

Format

```
.SYSTM
.GMCA
error return
normal return
```

Upon the normal return, AC1 contains the MCA unit number.

Possible errors

AC2	Mnemonic	Meaning
3	ERICD	Improper device code input to system call.
36	ERDNM	Device not in system (you did not specify an MCA at SYSGEN time in this RDOS system).

Using CLI Commands on MCA Lines

As described earlier, each MCA line has a filename. This means that you can use the MCA line filename in many CLI commands that take a filename argument. The only special requirement is that the CLI command be present in both receiver and transmitter, since no data transmission can occur without simultaneous receive and transmit requests.

Moreover, you can XFER (but you cannot LOAD or DUMP) disk files via MCA lines. Thus, in Figure 8-1, to transfer file ABC from CPU 4's disk CPU 2's disk, someone would type the following CLI commands on the appropriate consoles:

System 2

```
XFER MCAR:4 ABC)
```

CPU2 tells its MCA to transfer the contents of MCAR:4 to ABC on my disk. Because CPU2 is addressing a receive line, this is a receive request.

or

```
XFER MCAR:0 ABC)
```

CPU2 tells its MCA to Transfer any transmitter's input to ABC on my disk.

System 4

```
XFER ABC MCAT:2)
```

CPU4 tells its MCA to Transfer the contents of ABC on my disk to MCAT:2. Because CPU4 is addressing its transmitter, this is a transmit request.

When a CPU issues a CLI command over an MCA line, the CLI prompt won't return to its console until RDOS has executed the command (or, if the transmitter issues the request, until the transmitter has timed out).

Transmitting Copies of Operating Systems or Stand-Alone Programs

RDOS provides a bootstrap program, MCABOOT, which transfers and bootstraps a copy of an RDOS system to another unit's disk. Before sending the system, MCABOOT can either fully or partially initialize the receiver's disk. Alternatively, MCABOOT can send and bootstrap a copy of a stand-alone program to another unit's disk, provided that this program follows the conventions of programs which BOOT can load. (BOOT need not reside in the receiving unit's disk space.) As with other MCA data transfers, both receiver and transmitter must participate.

You execute the MCA bootstrap by issuing the CLI command MCABOOT. The transmitter and receiver must be on the same network (MCA or MCA1) for transmission to occur. An operator at the receiving CPU must have requested the transmission by placing 100007_8 (MCA) or 100047_8 (MCA1) in the receiver's data switches, and by pressing RESET followed by PROGRAM LOAD. The transmitting unit will wait for the receiver to request reception, but only up to the timeout period (655 seconds).

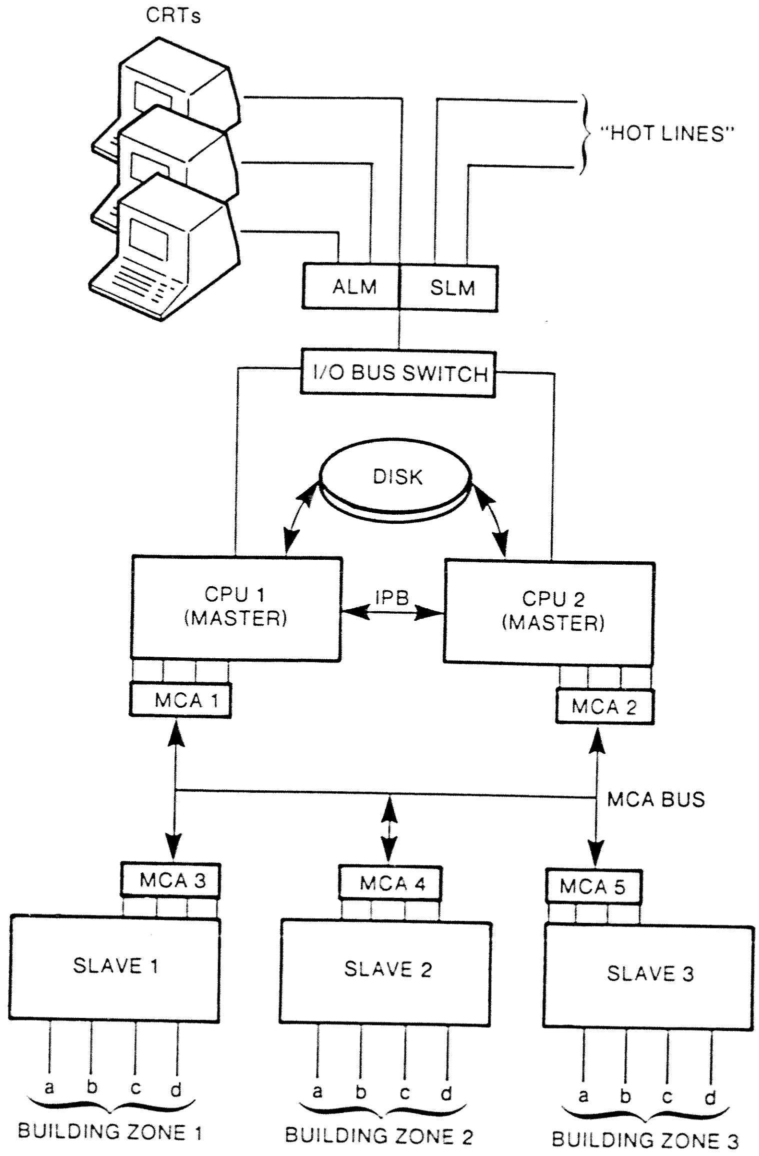
Multiprocessor System Illustration

The following example illustrates one application of a multiprocessor system. A large laboratory complex needs an automated system to control the environmental conditions within the complex, to keep track of the number of personnel at different locations, to monitor the complex for alarm conditions, and to alert key personnel if it cannot correct a condition. This system must be fail-safe, and can allow down-time for no longer than a few seconds.

Figure 8-2 suggests one configuration for this system. Two master CPUs, running under mapped RDOS, are connected via an IPB, so that each can act as a watchdog on the other's behavior, and take control if the other fails. The IPB also allows the CPUs to access common disk files. The masters access a common data base which contains, among other information, alarm messages and destinations to which they should go on an alert. This file space also contains a log of the current master's activity, so that if it should fail, the alternate master CPU would have a record of recent events.

The laboratory includes three vital zones, and there is a slave CPU to monitor and control conditions within each zone. Each slave can monitor and adjust both humidity and temperature. Additionally, each slave keeps track of the positions of personnel within each zone. Finally, each slave monitors its zone for alarm conditions; if they occur, it can take some remedial action to emergencies, e.g., it can activate a sprinkler system if it detects fire. Each slave computer does relatively simple things, and could run under RTOS, a core-resident compatible subset of RDOS.

Each slave has a data channel line through its MCA to each master computer (lines MCA1 through MCA6). This allows the current master to generate continuous status reports and transmit them to CRT monitors via the bus switch to an ALM. An SLM multiplexor connects "hot lines" to security guards and fire station personnel to alert them in an emergency.



a - temperature sensor and control
b - personnel monitor
c - humidity sensor and control
d - intrusion, fire, smoke alarm and control

SD-00553

Figure 8-2. Multiprocessor System Illustration

End of Chapter

Chapter 9
Tuning RDOS

This chapter describes tuning - a feature which allows an RDOS system to monitor its own performance, and suggest a more efficient configuration for any application. Concurrently, it explains some internal workings of RDOS. In this order, it explains:

- Tuning
- The data structures involved in tuning
- RDOS system overlays
- How tuning works
- Tuning system calls

During system generation, you tailor an RDOS system for a specific environment by answering SYSGEN questions. (For details on SYSGEN, read the manual *How to Load and Generate Your RDOS System.* This manual also has a practical section on tuning.) Your answers to the SYSGEN questions determine what features your RDOS system will have, and what peripheral hardware it will support. There are two tuning questions in SYSGEN, and your answers to these decide whether this RDOS system will have tuning at all, and how extensive the tuning function will be.

The tuning mechanism itself deals with certain software data structures, called *stacks, cells* and *buffers.* As it happens, SYSGEN asks questions about stacks, cells, and buffers. The tuning mechanism takes your answers to these questions and tests them as RDOS runs; it can then print a tuning report which allows you to decide on more efficient answers, or it can tell SYSGEN to modify your original answers during a new system generation.

The latter approach, called *self-tuning* can generate a moderately efficient version of RDOS for any application. During *self-tuning,* SYSGEN examines a previously-generated tuning report file and selects more appropriate responses to questions about buffers, stacks, and cells. You can direct a system to tune itself by including the name of the old SYSGEN dialog file, and the /T switch, in the SYSGEN command:

```
SYSGEN dialog-file/A tuning-file/T
```

SYSGEN examines the tuning file and attempts to generate a system that is more efficient for this application than the system that was running when the tuning file was recorded. During self-tuning, SYSGEN does not have a global view: it has only the tuning file to work from, hence it must make certain arbitrary decisions, such as the value of user memory to this given application. Thus SYSGEN can't determine the complete impact of tuning decisions upon any given application's efficiency. Nonetheless, it does an adequate job for applications which don't require maximum efficiency. Tuning file statistics by themselves are helpful, but other considerations are important too; in the final analysis, comparative timings of different system configurations provide the true measure of efficiency.

System Stacks, Cells, and Buffers

Before exploring tuning, we'll define the terms *system buffers, stacks,* and *cells.* RDOS is partially core-resident and partially disk-resident. This allows RDOS to have features found ordinarily only on larger operating systems, while the total memory-resident portion of RDOS in the system remains modest. Stacks, cells, and system buffers are all memory-resident parts of RDOS.

RDOS uses a system stack as a data base, to execute each concurrent .SYSTM call. The greater the number of outstanding .SYSTM requests, the more system stacks RDOS needs to service each request in parallel. For example, if two executing user tasks concurrently issue a .SYSTM call, two *system tasks* are then outstanding. To service both system tasks in parallel, RDOS would require two system stacks. At a single moment, RDOS will service only as many requests as it has available system stacks, in the order that these calls were made. System tasks are associated not only with .SYSTM calls, but also with I/O device requests and with spooling.

Each system task also requires a *cell,* to save state information, just as each user task has a Task Control Block. There is a fundamental difference between cells and TCBs, however: RDOS sometimes appropriates cells for temporary data storage, but it never uses TCBs for this purpose.

A large part of memory-resident RDOS is a collection of system *buffers,* which serve two functions. First, RDOS uses buffers to receive system overlays, which provide code not found in the resident portion of the system. Secondly, RDOS buffers all I/O except read/write block operations via system buffers.

RDOS requests and uses system stacks, buffers and cells dynamically, as resources. When it needs and cannot get any of these resources a *fault* occurs, it suspends the calling system task, and system operation suffers.

System Stack Requirements

The following guidelines will help you select the proper number of system stacks. RDOS needs stacks for Disk I/O, Spooling, and the concurrent execution of system calls, as follows:

System Task	Number of Stacks Required
Disk I/O	Two stacks if you will be running multitask programs, or foreground and background programs that need to issue disk I/O system calls concurrently (e.g., .OPEN, .INIT, .WRL).
Spooling	One stack
.SYSTM call	One stack for each user task that is to be able to execute a .SYSTM call (requiring the use of an I/O device) concurrently with other user tasks.

SYSGEN permits you to select from one to 10 (decimal) system stacks. If this RDOS system will run single task programs in a background - only environment, you need specify only one system stack. To spool output data, add another system stack. If you allocate only one stack, RDOS won't spool, and if you issue system spooling commands, it will treat them as no-ops. Likewise, if a system also has a foreground program active and you have defined only two stacks, no spooling will occur. We recommend at least 2 stacks for a single-ground system, and 3 for a dual-ground system (more for Extended BASIC).

To illustrate further, let's say that you have a background-only multitask program which spools to the line printer and performs disk I/O on only one channel at a time. This program requires the allocation of 3 system stacks: one stack for disk I/O, one for line printer output, and one for the spooler.

In general, you should allocate enough stacks to prevent .SYSTM calls issued to slow peripherals ($PTR, MTA, etc.) from interfering with .SYSTM calls necessary to support a real time environment. Each .RDL or .RDS to a non-multiplexed console requires a system stack until the read is completed.

Each system stack requires about 250_8 or 350_8 words depending on your computer; see *How to Load and Generate Your RDOS System* for exact figures. Add the stack total to the basic RDOS system memory requirements.

When the system attempts to allocate a stack and none is free, it suspends the calling task and control passes to whatever other system task is ready for execution; RDOS will attempt to allocate a stack for the suspended task at some future moment. Thus, the tuning report may indicate multiple unsuccessful stack requests for the same system task. The same is true of certain cell requests. However, all unsuccessful buffer and overlay requests, and most unsuccessful cell requests, cause the system task to wait until the appropriate resource becomes free.

System Cell Requirements

A system cell is a 20_8 -word control table which the system uses primarily to save system task state information. The optimum number of cells depends largely upon your system's application.

SYSGEN automatically allocates two cells for future read/write block operations, three cells for each stack, and two cells for an IPB (if you specified one); we recommend that you specify extra cells as follows: each active spool request requires two, and the IPB (if any) will run better with one or two extra cells.

Each active .SYSTM call also needs an extra cell.

Since one goal of tuning is to keep all peripheral devices active concurrently, you need not allocate a cell for every possible future concurrent .SYSTM call. For slow peripherals, a lack of cells can degrade the system's operation. Consider the following illustration in Figures 9-1 and 9-2. In Figure 9-1, this RDOS system contains three devices; a disk, a mag tape drive, and a line printer; it has nine cells. The program environment contains 20 user tasks, each one desiring the use of each of the three devices. It so happens that these tasks want to use different devices, hence the system runs efficiently. As each task issues an I/O request, RDOS enqueues its cell to that device so that when the device becomes free, the next task in line will eventually be able to use the device. Thus RDOS enqueues only nine system tasks for the devices (and stores 11 requests in a special system table, PTBL). Even though 11 requests are waiting in table PTBL, the system is running efficiently.

Notice the difference in Figure 9-2. This is the same system, but somehow nine ready tasks want to use the mag tape; these tasks monopolize the cell queue. Although up to 11 other tasks want to use the disk and line printer, they can't be readied until they receive a cell. RDOS will free cells one by one as the ready tasks finish with the mag tape; meanwhile, the other tasks stagnate in PTBL. There are too few cells for this program although there were enough for the same system in the first program. The waiting tasks cannot use the disk and line printer even though these devices are not busy, and the system is running inefficiently.

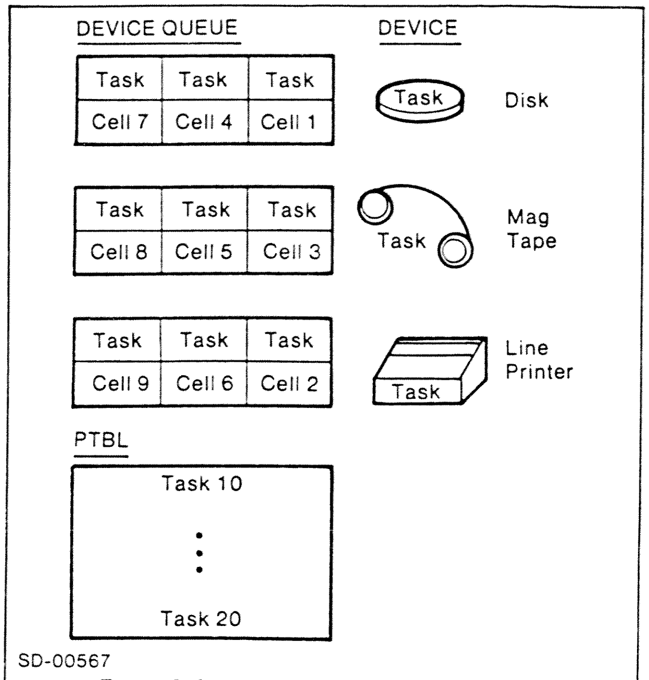


Figure 9-1. Adequate Cell Apportionment

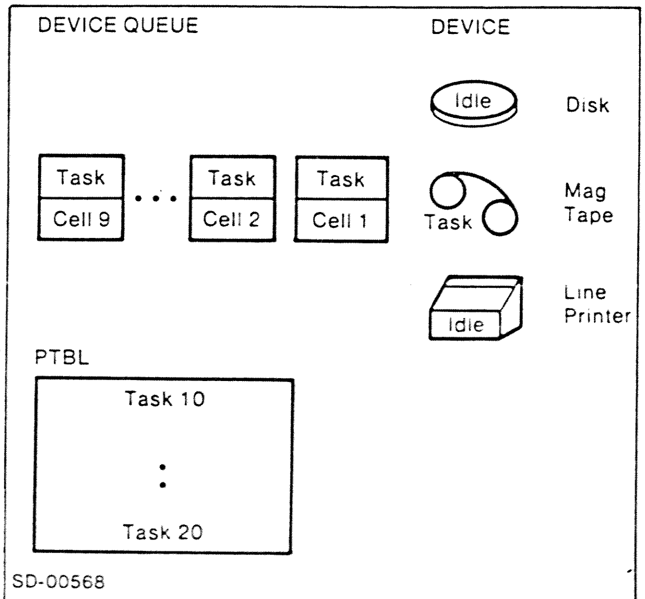


Figure 9-2. Inadequate Cell Apportionment

In the environements shown in both Figures 9-1 and 9-2, RDOS would report cell faults. However, additional cells would not improve system efficiency in Figure 9-1. Thus, you must supplement the fault information provided in the tuning report by timing your application programs to determine whether the reported faults really degrade system performance.

System Buffer Requirements

System buffers are portions of memory which RDOS allocates dynamically to receive either user data or system overlays. RDOS requires a minimum of two buffers per system stack or six buffers total, whichever is greater. SYSGEN automatically allocates this minimum; at SYSGEN time you can specify as many extra buffers as core memory will allow. Each system buffer requires 416_8 (274_{10} words. If any multiples of 274 words are available in the last 1024-word block of system space in mapped systems, RDOS will use these multiples for additional system buffers.

When RDOS needs a buffer, it flushes the contents of the oldest buffer that is not in use. However, if extra buffers are available, fewer get flushed and their contents can remain accessible to system memory. If your application favors having buffered data in core (for fast reaccess), or having many system overlays resident in core (for fast .SYSTM call execution), then you must specify extra buffers. Extra buffers increase system speed, but they reduce the total amount of memory available for your programs. Thus, the idea is to select enough system buffers to provide the speed you want, while leaving adequate memory for your programs.

RDOS requires some system buffers to receive system overlays. The following list describes each overlay, and the system calls or functions that it executes. Each overlay's number (octal) precedes its name in the list; you'll need this number to understand the tuning report, since the report doesn't include system overlay names.

Overlay Number	Name	Functions
0	DFRWS	Disk file .WRL, .RDR/.WRR, .RDS/.WRS.
1	DFRWS	Disk file .CHSTS, .RDL, .LINK, .RDL, .STAT.
2	UTIL1	Mag tape .GCHN, .GMEM, .SMEM; tape .MTDIO
3	CREATE	Starts file creation: .CONN, .CCONT, .CRAND, .CREAT
4	DELETE	Delete a file, a subdirectory or a secondary partition; .DELET.

Overlay Number	Name	Functions
5	FILSY	Maintains directories and searches for entries in them.
6	SOV1	Implements periodic rescheduling for .QTASK and QUE. Also implements the following system calls: .CHATR, .CHLAT, .FGND, .GCIN, .GCOUT, .GTATR, .GTOD,, .ODIS, .OEBL, and .STOD
7	SOV2	Checks file names for validity, interprets directory specifier prefixes, and unpacks file names into SYS.DR format.
11	SOV3	Processes disk file errors, reads disk core images.
11	SOV4	Opens files (.OPEN, .EOPEN, .ROPEN); .CLOSE, .RESET, and implements CLI CLEAR command.
12	DVINI	Initializes directories; .EQIV.
13	CRSFS	Creates a MAP.DR entry in SYS.DR, and creates peripheral device entries in SYS.DR after a full initialization.
14	RING1	Opens and closes character devices on level; writes messages to the console.
15	RING2	Console keyboard and reader .RDL/.RDS; .RDS, .GCHAR; system level character I/O (ACHR, WRS, PCH).
16	RING3	Performs system-level character I/O (ACHR, WRS, PCH).
17	SOV5	Performs housekeeping necessary to execute keyboard interrupt or .BREAK.
20	MTAIO	.INIT, .RLSE, .CLOSE; block-level reading and writing for magnetic tapes/cassette units.
21	MTAUC	.OPEN for magnetic tape/cassette units.
22	TUON	.TUON (turn tuning on).
23	CDROV	Card reader ASCII .RDL.
24	WDBLK	Completes the file creation activities originating in CREATE; withdraws a single block from MAP.DR.
25	SPOLR	Supports spooling.
26	CODER	Encodes and decodes 7-track magnetic tape; .SKPK, .SPDA, .SPEA.
27	SOV6	Writes a core image to disk.
30	SOV7	Continues disk core image read function started by SOV3.
31	SOV8	.EXEC, .EXFG, .EXBG.
32	SOV9	Resolves directory link entries; .EXEC, .EXFG, .EXBG.
33	SOV10	Continues directory resolution function of SOV9, .INIT, .RLSE.
34	SOV11	Determines size of a fixed-head disk for .INIT system call; .ICMN, .RDCM, .WRCM.
35	JEHOV	Creates an initial system directory.

Overlay Number	Name	Functions
36	SOV12	Continues the function performed by SOV5; creates file BREAK.SV and completes a program break caused either by .BREAK or console keyboard interrupt.
37	SOV13	Opens, closes a disk file; .UPDAT.
40	SOV14	.DIR, .RDOP/.WROP.
41	SOV15	.CDIR, .CPART.
42	SOV16	.IDEF, .DEBL/.DDIS.
43	FILS2	Preprocesses the deletion of partitions subdirectories; .RENAM.
44	SOV17	Finishes the housekeeping started by SOV5 for .EXEC/.RTN and keyboard interrupts; .IRMV.
45	SOV18	Produces an orderly shutdown upon a system release; .BOOT.
46	WDCBK	Withdraws a series of contiguous blocks from MAP.DR; creates an elemental MAP.DR for for DIVINI and SOV15.
47	SOV19	Determines size of a moving head disk during .INIT system call; does QTY open/close.
50	SOV20	Prepares program environment for a core-image load (mapped systems only).
51	SOV21	Provides MCA read/write sequential and and other MCA support functions.
52	SFTAB	This is a data overlay used to build pd peripheral device entries during a full system initialization.
53	SOV22	Continues the code begun in SOV18; .OVRP.
54	SOV23	Aborts a system process.
55	SOV24	Resolves spooling deadlocks; .GPOS/.SPOS, .GMCA; .OPEN for MCA.
56	SOV25	Completes the operation initiated by overlay WDCBK (46).
57	FSTAT	Provides support to other system overlays by getting and/or updating file status and by obtaining block addresses for disk I/O. Deposits a free block in MAP.DR.
60	DVRLS	Releases a directory; determines the DCT of a device for the spooling routines in CODER (26); .GDIR, .MDIR, .GSYS.
61	SOV26	.WRPR, .WREBL, .STMAP.
62	SOV27	Completes the execute functions starterted SOV8; continues the functions performed by by SOV3, SOV5 and SOV12; .RTN/.ERTN.
63	SOV28	.OVOPN; .MAPDF, .VMEM (mapped systems only).
64	TUNOV	.TUOFF (turn tuning off)
65	QTYOV	Provides QTY/ALM driver support.
66	SOV29	Replaces overlays in a .OL file.

How Tuning Works

After you have generated a system with tuning (having specified a number of stacks, cells and system buffers), you can turn tuning on, and start recording in the tuning file. If you find your system inefficient, you can examine the tuning report and reSYSGEN, specifying a different number of stacks, cells and/or buffers. As mentioned earlier, you can also have SYSGEN examine the tuning file, and modify the original answers to these questions. ReSYSGEN as often as you like, to generate one or more RDOS systems which run your application(s) well. You will cause a system failure, however, if you turn tuning on in a system before you have deleted the tuning file of a previous system with the same name.

As with many RDOS features, you can use either system calls or CLI commands to turn tuning on or off. You must, of course, have selected tuning and specified the kind of report you want at SYSGEN time as well as the type of tuning report you desire. SYSGEN automatically reserves extra buffers within the system for use by the report function. One buffer is required for the summary report; detailed reports require 3 buffers.

The CLI commands to turn tuning on and off are TUON and TUOFF; the command to display the contents of the tuning file is TPRINT. The corresponding system calls to turn tuning on and off are .TUON and .TUOFF. CLEARing the tuning file will not turn tuning off, nor will it affect the tuning report file. However, you must not delete this file while tuning is on. To produce a fresh tuning report, issue CLI commands TUOFF, RENAME or DELETE, and TUON.

When tuning is on, the tuning function accumulates the number of requests for stacks, cells, buffers, and system overlays. RDOS records this information in a disk file named *sysname.TU*, which resides in the master directory; sysname is the name of the current RDOS system. Additionally, RDOS also records the number of times it defaulted these requests because the resource was not available. You can then compute the ratio of requests to faults as an indication of the system efficiency.

Note that your program can access the tuning file by opening it, and then issuing system call .RDS for 2*TULEN bytes. (TULEN defines the number of words in the summary report).

The tuning report file is a contiguous disk file consisting of either one or three disk blocks depending on whether you requested an overlay report at SYSGEN. The first disk block contains the summary report. If you requested an overlay report, it follows on the next two disk blocks (Figure 9-3).

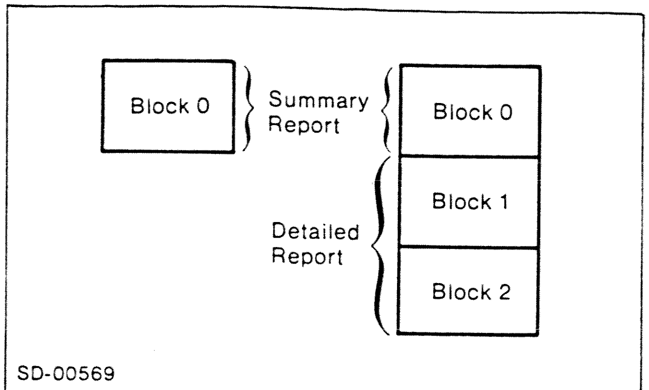
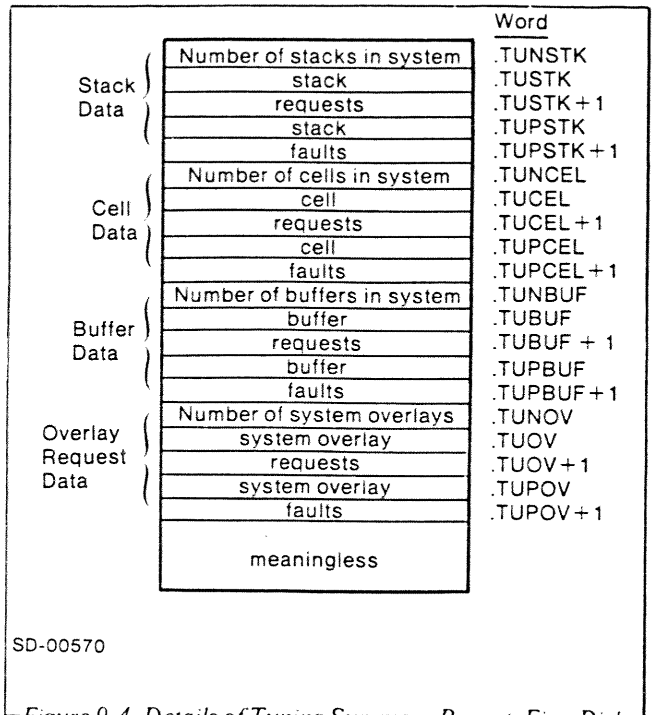


Figure 9-3. Disk Blocks of the Tuning File

The summary report contains four sections: one each for system stacks, cells, buffers, and system overlays. Each section in the summary is five 16-bit words long. The first word in each section lists the number of elements (stacks, buffers, etc.) in the system. The next two words are a double-precision integer count (2 16-bit words) of all requests for this element. The last two words are a double-precision integer count of faults, i.e., unsuccessful requests for the resource. Each double-precision count returns to zero upon overflow. The remaining words in the summary disk block are not meaningful.

Figure 9-4 shows the arrangement of information in the summary portion of the tuning report file. The named word displacements relative to the beginning of the file are defined in the file PARU.SR (user parameters supplied with your RDOS system; see Appendix B).



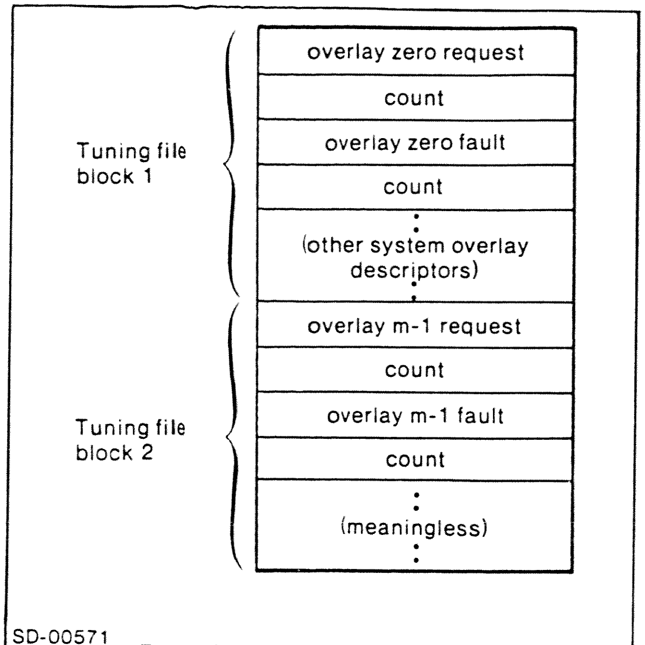
		Word
Stack Data	Number of stacks in system	.TUNSTK
	stack	.TUSTK
	requests	.TUSTK+1
	stack	.TUPSTK
	faults	.TUPSTK+1
Cell Data	Number of cells in system	.TUNCEL
	cell	.TUCEL
	requests	.TUCEL+1
	cell	.TUPCEL
	faults	.TUPCEL+1
Buffer Data	Number of buffers in system	.TUNBUF
	buffer	.TUBUF
	requests	.TUBUF + 1
	buffer	.TUPBUF
	faults	.TUPBUF+1
Overlay Request Data	Number of system overlays	.TUNOV
	system overlay	.TUOV
	requests	.TUOV+1
	system overlay	.TUPOV
	faults	.TUPOV+1
	meaningless	

Figure 9-4. Details of Tuning Summary Report, First Disk Block

The number of stacks and cells (displacements .TUNSTK and .TUNCEL) is the total of each in the system; the number of buffers (displacement .TUNBUF) is the total number of buffers, excluding tuning buffers. The buffer request count reflects the requests for data buffers and requests for buffers needed to receive system overlays. Also, as indicated earlier, multiple stack and cell faults can occur and be recorded for the same system task.

If you specified a detailed tuning report at SYSGEN time, then RDOS places it in the blocks immediately following the summary report in the tuning report file. The detailed report consists of a series of four-word descriptors, with one descriptor for each system overlay. Each descriptor contains a count of requests for a system overlay and a count of the number of requests requiring that the overlay be read from disk (because it was not then resident in memory). Each count is a double-precision integer; if an overflow occurs, RDOS returns the count to zero. The detailed report can list up to 128 separate system overlays. The counts of defined, but unused, overlays are set to zero. Figure 9-5 depicts the arrangement of information in the detailed report for a system with *m* overlays:

Each system overlay is described earlier in this chapter.



Tuning file block 1	overlay zero request
	count
	overlay zero fault
	count
	⋮ (other system overlay descriptors) ⋮
Tuning file block 2	overlay m-1 request
	count
	overlay m-1 fault
	count
	⋮ (meaningless) ⋮

Figure 9-5. Tuning Overlay Report

Start Recording in the Tuning File (.TUON)

This system command turns on the tuning mechanism, which reports system resources and faults in the tuning file. If the tuning report file does not exist, this command creates it as a contiguous file of either 1 or 3 blocks; the size depends upon your choice of report functions at SYSGEN time. RDOS names the file sysname.TU, and places it in the master directory; sysname is the name of the current RDOS system.

If the tuning file already exists, then new report information will be added to this file.

If the tuning report function is already on, this command is an effective no-op.

Required input

AC0 - Set to zero.

Format

```
.SYSTM
.TUON
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
2	ERICM	Illegal system command (tuning not SYSGENed).
27	ERSPC	Insufficient disk space to create tuning file.
46	ERICB	Insufficient number of free contiguous disk blocks available to create the tuning file.
101	ERDTO	Disk timeout occurred.

Stop Recording in the Tuning File (.TUOFF)

This system command halts the tuning report function until and unless you turn it back on with a .TUON command. This command does not delete the tuning file itself. It also releases any extra system buffers required by the tuning function, for use by the system.

If the tuning report function is already turned off, this call is an effective no-op.

Required input

None.

Format

```
.SYSTM
.TUOFF
error return
normal return
```

Possible errors

AC2	Mnemonic	Meaning
12	ERDLE	Tuning file was deleted before tuning was turned off.
101	ERDTO	Disk timeout occurred.

End of Chapter

Appendix A
RDOS Command and Error Summary

Table A-1 describes the required input to (or remarks on) the accumulators for each RDOS system or task call. n is the file's channel number, as assigned on the open. By default, after a task or system call, AC3 contains the User Stack Pointer (USP); To return the frame pointer, see *System and Task Calls* (Chapter 3). RDOS returns error codes, if any, in AC2. Task calls sometimes destroy ACs, as noted. SYSTM calls preserve ACs if they don't specifically return values.

Table A-1 Command Summary

Call	AC0	AC1	AC2
.ABORT	Destroyed.	Bits 8-15: task I.D. number.	
.AKILL[1]	Priority of tasks to be killed.		
.SYSTM .APPEND n	Byte pointer to file name.	Device characteristic mask (see .GTATR).	Channel number (if n = 77)
.ARDY[1]	Priority of tasks to be readied.		
.ASUSP[1]	Priority of tasks to be suspended.		
.SYSTM .BOOT[2]	Byte pointer to *primary-partition or specifier:filename.*		
.SYSTM[1] .BREAK[2]			
.SYSTM .CCONT	Byte pointer to file name.	Integer number of disk blocks.	
.SYSTM .CDIR	Byte directory to new directory name.		
.SYSTM .CHATR n	1B0: read-protect this file. 1B1: attribute-protect this file. 1B7: allow no link resolution. 1B9: user attribute. 1B10: user attribute. 1B14: make this file permanent. 1B15: write protect this file.		Channel number (if n = 77)
.SYSTM .CHLAT n	same as .CHATR		Channel number (if n = 77)

Footnotes

[1] no error return

[2] no normal return

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM CHSTS n	Starting address of 22_8 word area.		Channel number (if n = 77)
.SYSTM .CLOSE n			Channel number (if n = 77)
.SYSTM .CONN	Byte pointer to file name.	Integer number of disk blocks.	
.SYSTM .CRAND	Byte pointer to file name.		
.SYSTM .CPART	Byte pointer to secondary partition name.	Number of contiguous blocks (must exceed 60_8).	
.SYSTM .CREAT	Byte pointer to file name.		
.SYSTM .DDIS	Device code to be user-access disabled.		
.SYSTM .DEBL	Device code to be user-access enabled.		
.SYSTM .DELAY		Number of RTC ticks.	
.SYSTM .DELET	Byte pointer to file name.		
.SYSTM .DIR	Byte pointer to directory/directory device specifier.		
.DQTSK		Bits 8-15: task I.D. number.	(returned) Base address of released queue area.
.DRSCH[1]			
.SYSTM .DUCLK	Number of RTC ticks.	Address of user interrupt routine.	
.SYSTM .EOPEN n	Byte pointer to file name.	Characteristic disable mask (see .GTATR). 0 leaves characteristics unchanged.	Channel number (if n = 77)
.SYSTM .EQIV	Byte pointer to current disk or tape specifier.	Byte pointer to temporary specifier.	
.SYSTM .ERDB	Right byte: ext. memory block no. (0,1,2,or 3) Left byte: 256-word group no. (0,1,2,or 3)	Starting relative block no. in disk file.	Right byte: no. of 256-word blocks to be read[2] Left byte: channel no. (if n = 77)[2]

Footnotes

[1] no error return

[2] if error EOF, error code in right-byte, partial count in left byte.

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.ERSCH[1]			
.SYSTM .ERTN[2]			Data word to be passed to next higher level.
.SYSTM .EWRB	Right byte: extended memory block no. Left byte: 256-word group no. (0,1,2, or 3).	Starting relative block no. in disk file.	Right byte: no. of 256-word blocks to be written[3] Left byte: channel no. (if n = 77).[3]
.SYSTM .EXBG	Byte pointer to new BG program name.	0B1: new BG to have same priority as old BG. 1B1: new BG to have same priority as FG.	Optional message to new BG.
.SYSTM .EXEC	Byte pointer to save file name.	0: swap to user program. 1B0: chain to user program. 1: swap to debugger. 1B0 + 1: chain to debugger.	Message to new program.
.SYSTM .EXFG	Byte pointer to save file name.	0B1: FG to have over BG. 1B1: FG/BG equal priority. 0B15: pass control to save file. 1B15: pass control to debugger.	Passed to new program.
.SYSTM .FGND	(returned) 0	(returned) program level code: 1=BG level 0 ... 12 = FG level 4.	
.SYSTM .GCHAR	(returned) bits 9-15: character bits 0-8: cleared.		
.SYSTM			(returned) Free channel number.
.SYSTM .GCIN	Byte pointer to 6-byte area receiving the input console name.		
.SYSTM .GCOUT	Byte pointer to 6-byte area receiving the output console name.		
.SYSTM .GDAY	(returned) Day	(returned) Month	(returned) Year minus 1968

Footnotes

[1] no error return.

[2] no normal return.

[3] if error EOF, error code in right byte, partial count in left byte.

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM .GDIR	Byte pointer to 13_8 -byte area.		
.SYSTM .GHRZ	(returned) 0: no RTC 1: 10 HZ 2: 100 HZ 3: 1000 HZ 4: 60 HZ 5: 50 HZ		
.SYSTM .GMCA	MCA transmitter device code (6_8 or 46_8)	(returned) MCA unit number	
.SYSTM .GPOS n	(returned) High order portion of byte pointer.	(returned) Low order portion of byte pointer.	Channel number (if n = 77)
.SYSTM .GSYS	Byte pointer to 15_8 -byte area.		
.SYSTM .GTATR n	(returned) 1B0: read protected. 1B1: attribute protected. 1B2: save file 1B3: link entry* 1B4: partition* 1B5: directory file* 1B6: link resolution entry* 1B7: no link resolution allowed. 1B9: user attribute. 1B10: user attribute. 1B12: contiguous file* 1B13: random file* 1B14: permanent file 1B15: write-protected	(returned) MCA shares 0 and 15; see file PARU.SR 1B0: spoolable device. 1B1: 80-column card. 1B2: lower-to-uppercase. 1B3: form feed on open. 1B4: full word device. 1B6: LF after CR. 1B7: parity check/generation. 1B8: rubout after tab. 1B9: null after FF. 1B10: keyboard input. 1B11: TTY output. 1B12: no FF hardware. 1B13: operator intervention needed. 1B14: no TAB hardware. 1B15: leader/trailer.	Channel number (if n = 77)

Footnotes

* cannot be set by user

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM .GTOD	(returned) Seconds	(returned) Minutes	(returned) Hours (using a 24-hour clock).
.SYSTM .ICMN	Starting word address of communications area.	Size of area in words.	
.SYSTM .IDEF	Device code of user device.	DCT. (1B0 if data channel device is mapped systems). User power restart address if AC0 = 77_8	
.IDST[1]	0: ready 1: suspended by .SYSTM call. 2: suspended by .SUSP, .TIDS, .ASUSP. 3: waiting for .XMTW/.REC. 4: waiting for overlay node. 5: suspended by .SUSP, .ASUSP, or .TIDS and .SYSTM call. 6: suspended by .XMTW/.REC and .SUSP, .ASUSP, or .TIDS. 7: suspended by .ASUSP, .SUSP, or .TIDS and waiting for overlay node. 10: no such task exists.	bits 8-15: task I.D. number.	(returned) Base address of task's TCB.
.SYSTM .INIT	Byte pointer to directory/global device specifier.	-1: full (tape or disk) 0: partial	

Footnotes

[1] no error return

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM .INTAD			
.SYSTM .IOPC	Queue area address (0 if no RUN or QUE)	Right byte: max number of queue areas (0 if no RUN or QUE). Left byte: overlay channel no. (0 if right byte = 0).	Program Table Address (0 if no RUN or QUE).
.SYSTM .IRMV	Device code.		
.IXMT	Message address (destroyed).	Nonzero message (destroyed).	(destroyed)
.KILAD[1]	Address of kill-processing routine.		
.KILL[1, 2]			
.LEFD[1]	(contents lost upon return)		
.LEFE[1]	(contents lost upon return)		
.LEFS[1]	(returned) user status word 1B9 - LEF mode is enabled 0B9 - LEF mode is disabled		
.SYSTM .LINK	Byte pointer to link name.	0: link will be resolved in parent partition of link entry's residence. non-0: byte pointer is either to an alternate directory alias name or to an alias name string.	
.SYSTM .MAPDF	Number of blocks for extended addressing use.	Starting logical block number of window.	Size of window in 1K blocks.
.SYSTM .MDIR	Byte pointer to 13_8 byte area.		
.SYSTM .MEM	HMA	NMAX	
.SYSTM .MEMI	NMAX increment or decrement (2's complement)	(returned) new NMAX (after change)	

Footnotes

[1] no error return

[2] no normal return

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM .MTDIO n	Core data address, if a data transfer.	bit 0: 1, even parity; 0, odd parity. bits 1-3: 0, read (words). 1, rewind tape. 3, space forward. 4, space backwards. 5, write (words). 6, write EOF. 7, read device status word. bits 4-15: word or record count; if 0 on space command, position tape to new file if it is less than 4096 records away. (returned) number of words read/written or number of records spaced.	Channel number (if n = 77_8). Status word or system error code if error returns; status word if read status normal return. Returned: 1B0: error. 1B1: data late. 1B2: tape rewinding. 1B3: illegal command 1B4: high density or cassette if 1; low density if 0. 1B5: parity error. 1B6: end of tape. 1B7: end of file. 1B8: tape at load point 1B9: 9-track or cassette if 1; 7-track if 0. 1B10: bad tape; write failure. 1B11: send clock (0 if cassette). 1B12: first character (0 if cassette). 1B13: write-protected or write-locked. 1B14: odd character (0 if cassette). 1B15: unit ready.
.SYSTM .MTOPD n	Byte pointer to tape global specifier.	Characteristic inhibit mask (see .GTATR)	Channel number (if n = 77)
.SYSTM .ODIS			
.SYSTM .OEBL			
.SYSTM .OPEN n	Byte pointer to file name.	Characteristic inhibit mask (see .GTATR); 0 leaves previous characteristic unchanged. 0 for MCA, or 01 to specify your own MCAT retry timeout.	Channel number (if n = 77)

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.OVEX[3]	Bits 0-7: node number Bits 8-15: overlay number.		Return address.
.OVKIL[4]	Bits 0-7: node number Bits 8-15: overlay number.		
.SYSTM .OVLOD n	Bits 0-7: node number. Bits 8-15: overlay number.	-1: unconditional 0: conditional	Channel number (if n = 77)
.SYSTM .OVOPN n	Byte pointer to overlay file name (with .OL extension)		Channel number (if n = 77)
.OVREL	Bits 0-7: node number. Bits 8-15: overlay number.		
.SYSTM OVRP	Byte pointer to overlay replacement file name (.OR).	Byte pointer to overlay file name (.OL).	
.SYSTM .PCHAR	Bits 9-15: character; Bits 0-8: ignored.		
.PRI[1]	Bits 8-15: new task priority.		
.QTSK			Address of User Task Queue table.

Footnotes

[1] no error return

[2] if error EREOF, error code in bits 8-15, partial read count in bits 0-7.

[3] normal return through AC2

[4] no normal return

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM .RDB n	Starting core address to receive data.	Starting disk relative block number.	Bits 0-7: number of blocks to be read[2] Bits 8-15: channel number (if n = 77)[2]
.SYSTM .RDCM	Word address to read into.	Offset into communications area.	Word count.
.SYSTM .RDL n	Byte pointer to user core area.	(returned) Read byte count (including terminator).	Channel number (if n = 77)
.SYSTM .RDOP	Byte pointer to message area.	(returned) Byte count.	
.SYSTM .RDR n	Core address to receive record.	Record number.	Channel number (if n = 77)
.SYSTM .RDS n	Byte pointer to core area. (Must be even for MCA).	Number of bytes to be read (if EOF detected, partial byte count returned).	Channel number (if n = 77)
.SYSTM .RDSW	(returned) Console with switch position.		
.REC[1]	Message address.	Message.	
.REMAP	Destroyed.	Destroyed. Left byte: starting relative block number in map. Right byte: starting relative block number of window.	Destroyed. Number of 1K blocks to be remapped.
.SYSTM .RENAM	Byte pointer to old name.	Byte pointer to new name.	
.SYSTM .RESET			
.SYSTM .RLSE	Byte pointer to directory or global device specifier.		
.SYSTM .ROPEN n	Byte pointer to file name.	Characteristic inhibit mask (see .GTATR); 0 preserves characteristics without change. For MCA, see .OPEN.	Channel number (if n = 77)
.SYSTM .RSTAT	Byte pointer to file name string.	Starting address of 22. word area.	
.SYSTM .RTN[1]			
SYSTM RUCLK			

Footnotes

[1] no error return

[2] no normal return. If error EREOF, error code in bits 8-15, partial read count in bits 0-7

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM .SDAY	Day	Month	Year minus 1968
.SMSK[1]	Lost	New interrupt mask to be ORed with old mask.	Lost
.SYSTM .SPDA	Byte pointer to device name.		
.SYSTM .SPEA	Byte pointer to device name.		
.SYSTM .SPKL	Byte pointer to device name.		
.SYSTM .SPOS n	High order portion of byte pointer.	Low order portion of byte pointer.	Channel number (if n = 77)
.SYSTM .STAT	Byte pointer to file name string.	Starting address of 22_8 word area.	
.SYSTM .STMAP	Device code.	Starting user address of device buffer. (Logical address of device buffer is returned.)	
.SYSTM .STOD	Seconds	Minutes	Hours
.SUSP[1]			
.TASK	Left byte: task I.D. number Right byte: task priority.	New task entry point address.	Contents passed to new task.
.TIDK		Right byte: task I.D. number.	
.TIDP	Bits 8-15: new priority	Right byte: task I.D. number.	
.TIDR		Right byte: task I.D. number.	
.TIDS		Right byte: task I.D. number.	
.TOVLD	Bits 0-7: area number Bits 8-15: overlay number.	-1: unconditional load. 0: conditional load.	Channel number on which overlay file was .OVOPNed.
.TRDOP	Byte pointer to message area (must be even).	(returned) Byte count	

Footnotes

[1] no error return

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM .TUOFF			
.SYSTM .TUON	0		
.TWROP	Byte pointer to message area.	-1 to suppress task ID number.	
.UCEX [1, 2, 3]		Any nonzero value to force rescheduling.	
.UIEX [1, 2, 3]		Any nonzero value to force rescheduling.	Unmapped: value upon the call = return address Mapped: unimportant
.SYSTM .ULNK	Byte pointer to link entry name.		
.SYSTM .UPDAT n			Channel number (if n = 77)
.UPEX [1, 2, 3]			
.SYSTM .VMEM	(returned) Number of available blocks.		
.SYSTM .WRB n	Starting memory address.	Starting relative block number.	Left byte: number of disk blocks[4] Right byte: channel number (if n = 77)[4]
.SYSTM .WRCM	Word address of message.	Offset into communication area.	Word count.
.SYSTM .WREBL	Starting address of series.	Ending address of series.	
.SYSTM .WRL n	Byte pointer to core buffer.	Write byte count, including terminator, returned at end of write.	Channel number (if n = 77)
.SYSTM .WROP	Byte pointer to text string.		

Footnotes

[1] no error return

[2] no normal return

[3] unmapped: on the interrupt, AC3 contained the return address. You must restore AC3 to this value before you issue this call.

[4] if error ERSPC, error code in right byte, partial write count in left byte.

Table A-1. Command Summary (continued)

Call	AC0	AC1	AC2
.SYSTM .WRPR	Starting address of 1K block series.	Ending address of 1K block series.	
.SYSTM .WRR n	Core address of record.	Record number.	Channel number (if n = 77)
.SYSTM .WRS n	Byte pointer to core buffer (must be even for MCA)	Number of bytes to be written.	Right byte: Channel number (if n = 77) Left byte: No. of MCA retries (each retry takes 200 milliseconds)
XMT	Message address.	Message must be (nonzero).	
XMTW	Message address.	Message must be (nonzero).	

Error Message Summary

Applicable commands are arranged alphabetically in columns, in descending order.

Table A-2. Error Message Summary

Code	Mnemonic	Meaning	Applicable Commands			
0	ERFNO	Illegal channel number.	.APPEND .CHATR .CHSTS .CHLAT .CLOSE .EOPEN .ERDB	.EWRB .GPOS .GTATR .MTDIO .MTOPD .OPEN .OVLOD	.OVOPN .RDB .RDL .RDR .RDS .ROPEN .SPOS	.WRB .WRL .WRR .WRS .UPDAT
1	ERFNM	Illegal file name.	.APPEND .BOOT .CCONT .CDIR .CONN .CPAR .CRAND	.CREAT .DELET .DIR .EOPEN .EXBG .EXEC .EXFG .INIT	.LINK .MTOPD .OPEN .OVOPN .OVRP .RENAM .ROPEN	.RLSE .RSTAT .SPDA .SPEA .SPKL .STAT .UNLK
2	ERICM	Illegal system command.	.TUON; any unimplemented call.			
3	ERICD	Illegal command for device.	.APPEND .ERDB .EWRB .GCHAR .GMCA .MTDIO	.MTOPD .PCHAR .RDB	.RDS .RDL .SPDA .SPEA	.SPKL .WRB .WRL .WRS
4	ERSV1	File requires the Save attribute and the ranDom characteristic.	.ERDB .EWRB	.EXBG .EXEC .EXFG	.RDB	.WRB
6	EREOF	End of file	.ERDB .EWRB	.OVLOD .OVOPN .RDB	.RDL .RDR .RDS .WRB	.WRL .WRR .WRS
7	ERRPR	Attempt to read a read-protected file.	.ERDB .EWRB	.OVLOD .OVOPN	.RDB .RDL	.RDR .RDS
10	ERWPR	Attempt to write a write-protected file.	.EWRB .INIT	.WRB .WRL	.WRR	.WRS
11	ERCRE	Attempt to create an existent file.	.CCONT .CDIR	.CONN .CPAR	.CRAND .CREAT	.LINK .RENAM
12	ERDLE	Attempt to reference a nonexistent file.	.APPEND .BOOT .DELET .DIR	.EOPEN .EXBG .EXEC .EXFG .INIT	.MTOPD .OPEN .OVOPN .OVRP .ROPEN	.RSTAT .STAT .TUOFF .ULNK
13	ERDE1	Attempt to alter a permanent file.	.DELET	.RENAM	.ULNK	
14	ERCHA	Illegal attempt to change file attributes.	.CHATR	.CHLAT		

Table A-2. Error Message Summary (continued)

Code	Mnemonic	Meaning	Applicable Commands			
15	ERFOP	Attempt to reference an unopened file.	.CHATR .CHLAT .CHSTS .CLOSE .ERDB	.EWRB .GPOS .GTATR .MTDIO .OVLOD	.RDB .RDL .RDR .RDS .SPOS	.UPDAT .WRB .WRS .WRR .WRL
16	ERFUE	Fatal utility error.	.EXEC (argument to .ERTN).			
17	EREXQ	Execute CLI.CM on return to CLI. (This is not really an error, but an instruction.)	.ERTN	.EXEC	.RTN	
20	ERNUL	Invisible error code.	.EXEC (argument to .ERTN)			
21	ERUFT	Attempt to use a channel already in use.	.APPEND .EOPEN	.GCHN .MTOPD	.OPEN .OVOPN	
22	ERLLI	Line limit exceeded on read or write line.	.RDL	.WRL		
23	ERRTN	Attempt to restore a nonexistent image.	.BOOT			
24	ERPAR	Parity error on read line. Magnetic tape parity. (Often caused by dirty heads.)	.RDL	.RDS		
25	ERCM3	Trying to push too many levels.	.EXBG	.EXEC		
26	ERMEM	Attempt to allocate more memory than available.	.EXBG .EXEC	.EXFG .MAPDF	.MEMI .OVOPN	
27	ERSPC	Out of disk space. Magnetic tape - EOT	.BREAK .CCONT .CDIR .CPAR .CRAND	.CREAT .DIR .EWDB .INIT	.OPEN .OVRP .LINK .MTOPD	.TUON .WRL .WRR .WRS
30	ERFIL	File read error. Magnetic tape - bad tape - odd count. Often caused by dirty heads.	.ERDB .OVLOD	.OVOPN .RDB	.RDR	.RDS
31	ERSEL	Unit improperly selected.	.APPEND .DIR	.EOPEN .INIT	.OPEN .MTOPD	.ROPEN .RLSE
32	ERADR	Illegal starting address.	.EXEC .EXFG	.MAPDF .REMAP	.WREBL	.WRPR
33	ERRD	Attempt to read into system area.	.CHSTS .GCIN .GCOUT	.GDIR .GSYS .MDIR	.RDB .RDL .RDR	.RDS .RSTAT .STAT
34	ERDIO	Attempt to perform direct block I/O on a sequentially organized file.	.ERDB	.EWRB	.RDB	.WRB
35	ERDIR	Files specified on different directories.	.RENAM			

Table A-2. Error Message Summary (continued)

Code	Mnemonic	Meaning	Applicable Commands			
36	ERDNM	Device not in system or illegal device code.	.APPEND .DIR .DEBL .EOPEN .GMCA	.IDEF .INIT .IRMV .MTOPD	.OPEN .RLSE .ROPEN .RSTAT	.SPDA .SPEA .SPKL .STAT
37	EROVN	Illegal overlay number.	.OVEX .OVKIL	.OVLOD	.OVREL	.TOVLOD
40	EROVA	File not accessible by direct (free form) I/O.	.ERDB .EWRB	.MTDIO .OVLOD	.OVOPN .RDB	.TOVLD .WRB
41	ERTIM	Attempt to set illegal time or date.	.SDAY	.STOD		
42	ERNOT	Out of TCBs.	.IOPC	.TASK	.TWROP	
43	ERXMT	Message address is already in use.	.IXMT	.XMT	.XMTW	
45	ERIBS	Interrupt device code in use.	.DUCLK	.IDEF	.RUCLK	
46	ERICB	Insufficient number of free contiguous disk blocks to create file.	.CCONT	.CONN	.CPAR	.TUON
47	ERSIM	Duplicate read or duplicate write to mux line.	.RDL	.RDS	.WRL	.WRS
50	ERQTS	Illegal information in task queue table.	.QTSK			
51	ERNMD	Attempt to open too many devices or directories.	.DIR	.INIT		
52	ERIDS	Illegal directory specifier.	.DIR	.INIT		
53	ERDSN	Directory specifier unknown.	.APPEND .BOOT .CCONT .CDIR .CONN .CPAR	.CRAND .CREAT .DELET .DIR .EOPEN .EQIV	.EXBG .EXEC .EXFG .LINK .MTOPD .OPEN	.OVOPN .RENAM .ROPEN .RSTAT .STAT .ULNK
54	ER2DS	Partition is too small.	.CPAR			
55	ERDDE	Directory depth exceeded.	.CDIR	.CPAR		
56	ERDIU	Directory in use.	.INIT	.DELET	.EQIV	.RLSE
57	ERLDR	Link depth exceeded.	.APPEND .CCONT .CDIR .CONN .CPAR .CRAND	.CREAT .DELET .DIR .EOPEN .EXBG .EXEC	.INIT .LINK .MTOPD .OPEN .OVOPN .OVRP	.RENAM .ROPEN .RSTAT .STAT .ULNK

Table A-2. Error Message Summary (continued)

Code	Mnemonic	Meaning	Applicable Commands			
60	ERFIU	File is in use.	.APPEND .BREAK	.DELET .EOPEN	.OPEN	.RENAM
61	ERTID	Task I.D. error.	.ABORT .DQTSK	.TASK .TIDK	.TIDP .TIDR	.TIDS
62	ERCMS	Communications area size error.	.ICMN	.RDCM	.WRCM	
63	ERCUS	Communications usage error.	.RDCM	.WRCM		
64	ERSCP	File position error.	.SPOS			
65	ERDCH	Insufficient room in data channel map.	.IDEF			
66	ERDNI	Directory/device not initialized.	.APPEND .BOOT .CCONT .CDIR .CONN .CPAR	.CRAND .CREAT .DELET .DIR .EOPEN .EXBG	.EXEC .EXFG .LINK .MTOPD .OPEN .OVOPN	.OVRP .RENAM .ROPEN .RSTAT .STAT .ULNK
67	ERNDD	No default directory.				
70	ERFGE	Foreground already exists.	.EXFG	.ECLR	.SMEM	
71	ERMPT	Error in partition set.				
72	EROPD	Released directory in use by other program.	.RLSE	.BOOT		
73	ERUSZ	Not enough room for UFTs within USTCH.	.EXEC	.EXFG	.EXBG	
74	ERMPR	Address outside address space (in a mapped system only).	.APPEND .BOOT .CCONT .CDIR .CHSTS .CONN .CPAR .CRAND .CREAT .DELET .DIR .DUCLK .EOPEN .EQIV .ERDB	.EWRB .EXBG .EXEC .EXFG .GCIN .GCOUT .GDIR .GSYS .ICMN .IDEF .INIT .MDIR .MEMI .MTDIO .MTOPD	.OPEN .OVOPN .OVRP .RDB .RDCM .RDL .RDOP .RDR .RDS .RENAM .RLSE .ROPEN .RSTAT .SPDA .SPEA	.SPKL .STAT .STMAP .TRDOP .TWROP .UNLK .WRB .WRCM .WREBL .WRL .WROP .WRPR .WRR .WRS
75	ERNLE	Attempt to delete an entry lacking the link characteristic.	.ULNK			
76	ERNTE	Background program is not checkpointable.	.EXBG			

Table A-2. Error Message Summary (continued)

Code	Mnemonic	Meaning	Applicable Commands			
77	ERSDE	Error detected in SYS.DR.	.INIT	.CREATE .RENAME .CPART .CDIR .CCONT .CRAND .LINK	If error occurs when directory has to be extended.	
100	ERMDE	Error detected in MAP.DR.	.BREAK	.DELET		
101	ERDTO	Device timeout.	.APPEND .BREAK .CCONT .CDIR .CHATR .CHLAT .CHSTS .CONN .CPAR .CRAND .CREAT	.DELET .DIR .EOPEN .ERDB .EWRB .EXBG .EXEC .EXFG .GTATR .LINK .OPEN	.OVLOD .OVOPN .OVRP .RDB .RDR .RDS .RENAM .RESET .RLSE .ROPEN .RSTAT	.STAT .TOVLD .TUOFF .TUON .UNLK .UPDAT .WRB .WRL .WRR .WRS
102	ERENA	Link not allowed.	.APPEND .DELETE	.EOPEN .EXEC	.INIT .OPEN	.OVOPN .ROPEN
103	ERMCA	No complementary MCA request.	.RDS	.WRL	.WRS	
104	ERSRR	Short MCA receive request.	.RDS	.WRL	.WRS	
105	ERSDL	System deadlock (RDOS is out of buffers).				
106	ERCLO	I/O terminated by a channel close.	.RDL .RDS	.WRL	.WRS	.RDS
107	ERSFA	Spool file is active.	.ABORT	.BOOT		
110	ERABT	Task not found for abort.	.ABORT			
111	ERDOP	Attempt to open a magnetic tape or cassette unit that is already open.	.APPEND .EOPEN	.OPEN	.MTOPD	.ROPEN
112	EROVF	System stack overflow (the current system command is aborted).	.INIT	.DIR		
113	ERNMC	No outstanding receive request by an MCA device.	.WRS			
114	ERNIR	Attempt to initialize or release a tape unit with a currently open file.	.INIT	.RLSE		
115	ERXMZ	Attempt to transmit a zero-word message.	.XMT	.XMTW	.IXMT	
116	ERCANT	Gross input error, such as ECLIPSE code on a NOVA, or lowercase ASCII characters.				

Table A-2. Error Message Summary (continued)

Code	Mnemonic	Meaning	Applicable Commands			
117	ERQOV	.TOVL not loaded for overlay task.	.QTASK			
120	EROPM	Operator messages not specified at SYSGEN time.	.IOPC .RDOP	.TRDOP	.WROP	.TWROP
121	ERFMT	Disk format error. If it recurs, DUMP disk and run DKINIT.SV.	.INIT	.DIR		
122	ERBAD	Disk has invalid bad block table. Run DKINIT.SV.	.INIT	.DIR		
123	ERBSPC	Insufficient space in core for bad block pool.	.INIT	.DIR		
124	ERZCB	Attempt to create a contiguous file of zero length.	.CCONT	.CONN		
125	ERNSE	Program is not swappable.	.EXEC			
126	ERBLT	Blank tape.	.OPEN	.MTOPD		
127	ERRDY	Line not ready, modem's DSR is low (multiplexors only).	.RDL	.RDS	.WRL	.WRS
130	ERINT	Console interrupt received (mux only).	.RDL	.RDS		
131	EROVR	Hardware overrun error (mux only).	.RDL	.RDS		
132	ERFRM	Hardware framing error (mux only).	.RDL	.RDS		
133	ERSPT	Too may framing errors (DOS only, not RDOS).				
134	ERPWC	Previous .WCHAR outstanding (returned from .WCHAR only).				

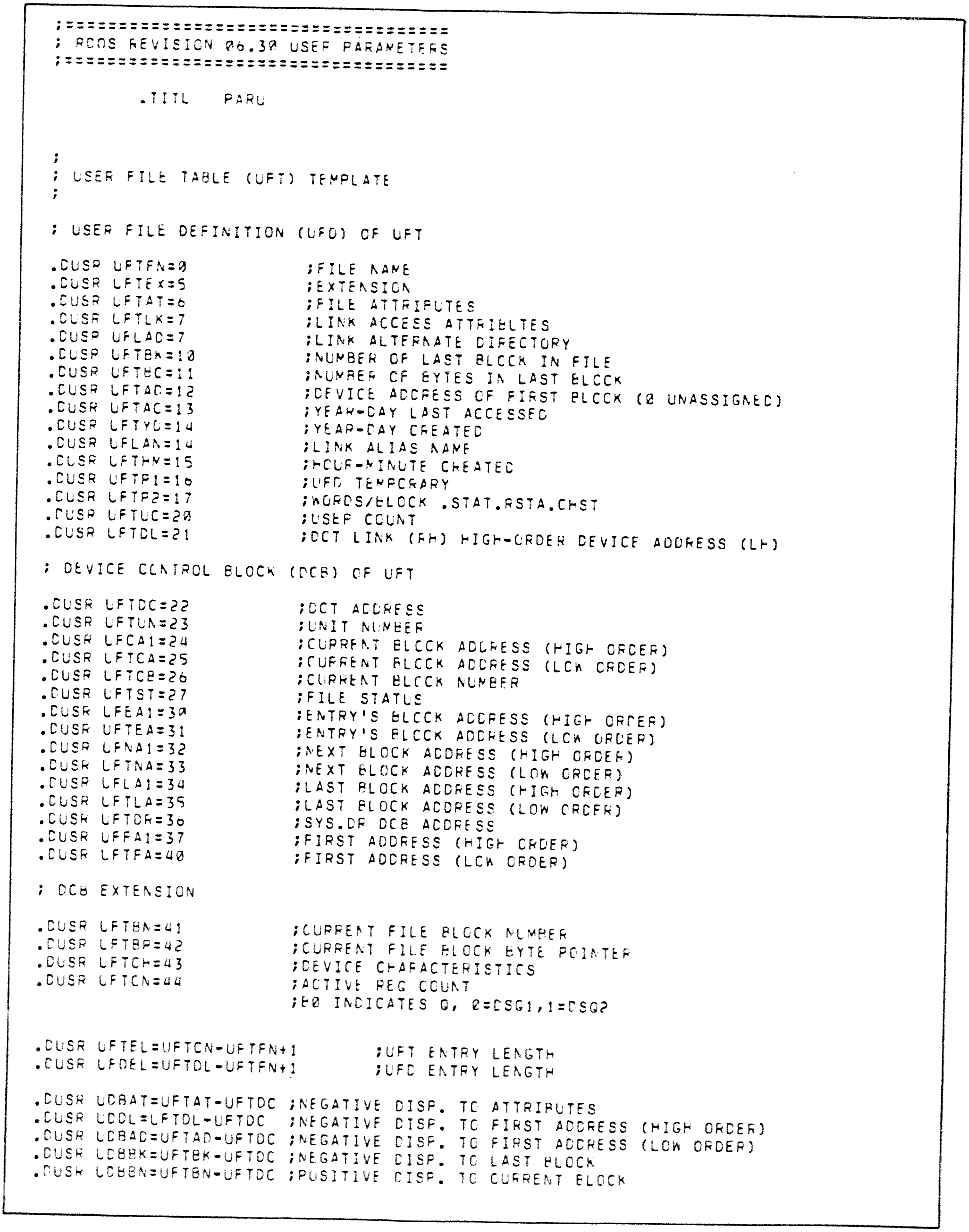
End of Appendix

Appendix B
User Parameters

This appendix lists file PARU.SR, which describes all RDOS user parameters. These parameters define important system calls, task calls, and mnemonics for user programs. PARU.SR was delivered with your RDOS system. File PARS.SR contains all *system* parameters.

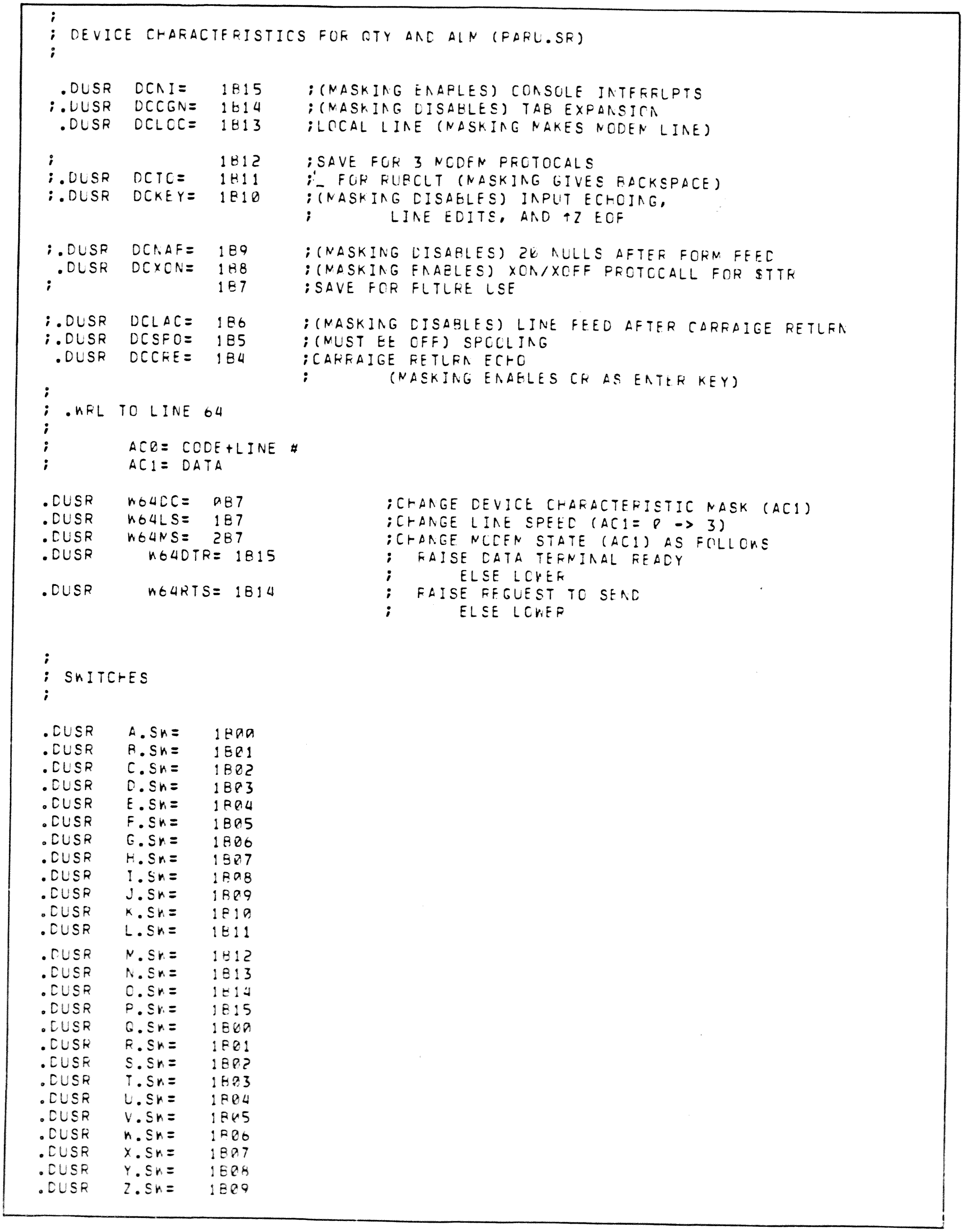
During SYSGEN, these files were loaded into the master directory.

An assembler cross-reference listing follows the parameter listing. Use this cross-reference to find individual parameters. The numbers in the cross-reference indicate *listing* pages, not appendix pages. For example, parameter UFTCN has entries 1/56 and 2/05; these indicate listing page 1, line 56 and listing page 2, line 5, respectively.

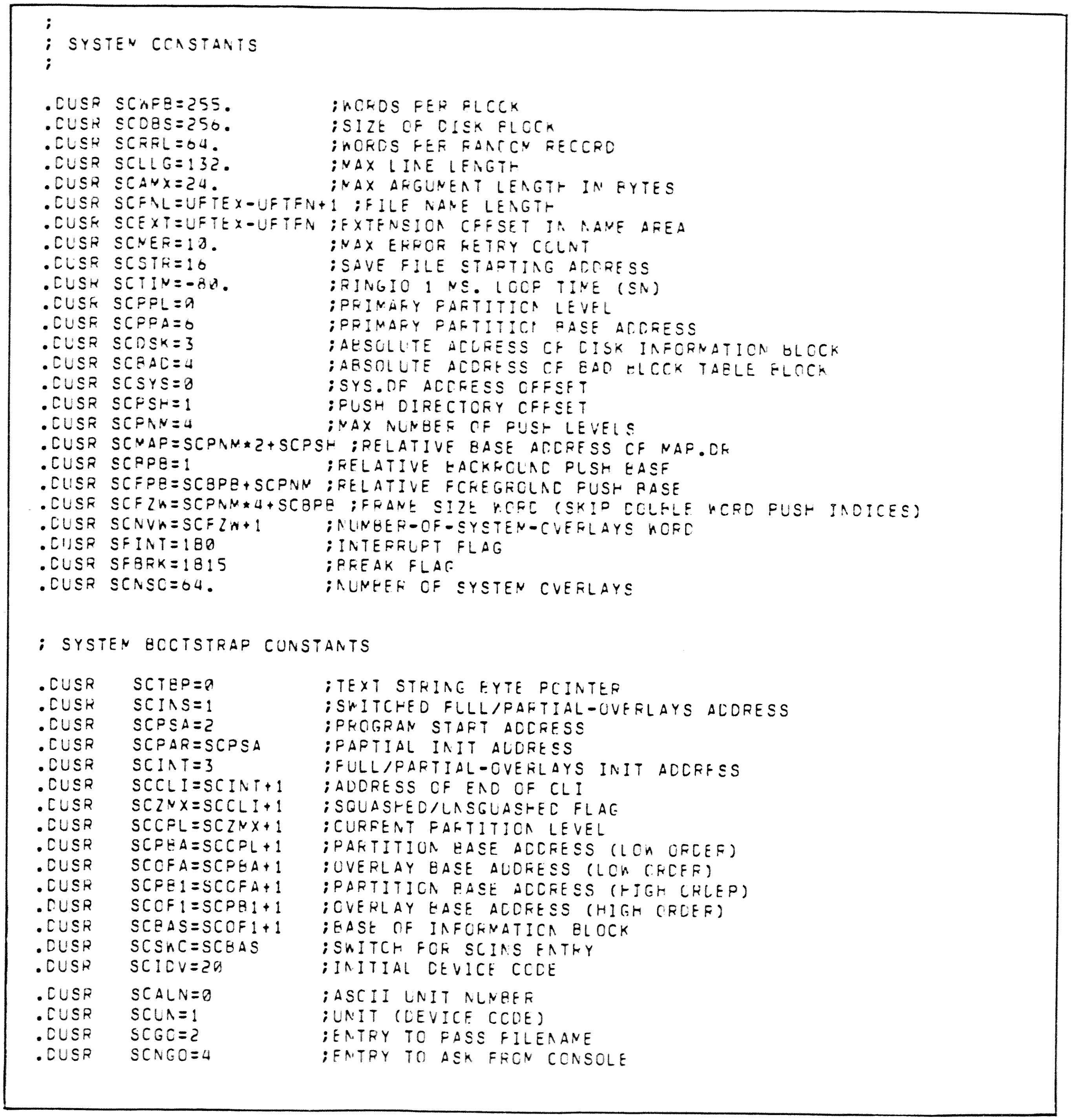


```
;==================================
; RDOS REVISION 06.30 USER PARAMETERS
;==================================

        .TITL   PARU


;
; USER FILE TABLE (UFT) TEMPLATE
;

; USER FILE DEFINITION (UFD) OF UFT

.DUSR UFTFN=0           ;FILE NAME
.DUSR UFTEX=5           ;EXTENSION
.DUSR UFTAT=6           ;FILE ATTRIBUTES
.DUSR UFTLK=7           ;LINK ACCESS ATTRIBUTES
.DUSR UFLAD=7           ;LINK ALTERNATE DIRECTORY
.DUSR UFTBK=10          ;NUMBER OF LAST BLOCK IN FILE
.DUSR UFTBC=11          ;NUMBER OF BYTES IN LAST BLOCK
.DUSR UFTAD=12          ;DEVICE ADDRESS OF FIRST BLOCK (0 UNASSIGNED)
.DUSR UFTAC=13          ;YEAR-DAY LAST ACCESSED
.DUSR UFTYD=14          ;YEAR-DAY CREATED
.DUSR UFLAN=14          ;LINK ALIAS NAME
.DUSR UFTHM=15          ;HOUR-MINUTE CREATED
.DUSR UFTP1=16          ;UFD TEMPORARY
.DUSR UFTP2=17          ;WORDS/BLOCK .STAT.RSTA.CHST
.DUSR UFTUC=20          ;USER COUNT
.DUSR UFTDL=21          ;DCT LINK (RH) HIGH-ORDER DEVICE ADDRESS (LH)

; DEVICE CONTROL BLOCK (DCB) OF UFT

.DUSR UFTDC=22          ;DCT ADDRESS
.DUSR UFTUN=23          ;UNIT NUMBER
.DUSR UFCA1=24          ;CURRENT BLOCK ADDRESS (HIGH ORDER)
.DUSR UFTCA=25          ;CURRENT BLOCK ADDRESS (LOW ORDER)
.DUSR UFTCB=26          ;CURRENT BLOCK NUMBER
.DUSR UFTST=27          ;FILE STATUS
.DUSR UFEA1=30          ;ENTRY'S BLOCK ADDRESS (HIGH ORDER)
.DUSR UFTEA=31          ;ENTRY'S BLOCK ADDRESS (LOW ORDER)
.DUSR UFNA1=32          ;NEXT BLOCK ADDRESS (HIGH ORDER)
.DUSR UFTNA=33          ;NEXT BLOCK ADDRESS (LOW ORDER)
.DUSR UFLA1=34          ;LAST BLOCK ADDRESS (HIGH ORDER)
.DUSR UFTLA=35          ;LAST BLOCK ADDRESS (LOW ORDER)
.DUSR UFTDR=36          ;SYS.DR DCB ADDRESS
.DUSR UFFA1=37          ;FIRST ADDRESS (HIGH ORDER)
.DUSR UFTFA=40          ;FIRST ADDRESS (LOW ORDER)

; DCB EXTENSION

.DUSR UFTBN=41          ;CURRENT FILE BLOCK NUMBER
.DUSR UFTBP=42          ;CURRENT FILE BLOCK BYTE POINTER
.DUSR UFTCH=43          ;DEVICE CHARACTERISTICS
.DUSR UFTCN=44          ;ACTIVE REG COUNT
                        ;B0 INDICATES Q, 0=DSG1,1=DSG2

.DUSR UFTEL=UFTCN-UFTFN+1         ;UFT ENTRY LENGTH
.DUSR UFDEL=UFTDL-UFTFN+1         ;UFD ENTRY LENGTH

.DUSR UDBAT=UFTAT-UFTDC ;NEGATIVE DISP. TO ATTRIBUTES
.DUSR UDDL=UFTDL-UFTDC  ;NEGATIVE DISP. TO FIRST ADDRESS (HIGH ORDER)
.DUSR UDBAD=UFTAD-UFTDC ;NEGATIVE DISP. TO FIRST ADDRESS (LOW ORDER)
.DUSR UDBBK=UFTBK-UFTDC ;NEGATIVE DISP. TO LAST BLOCK
.DUSR UDBBN=UFTBN-UFTDC ;POSITIVE DISP. TO CURRENT BLOCK
```

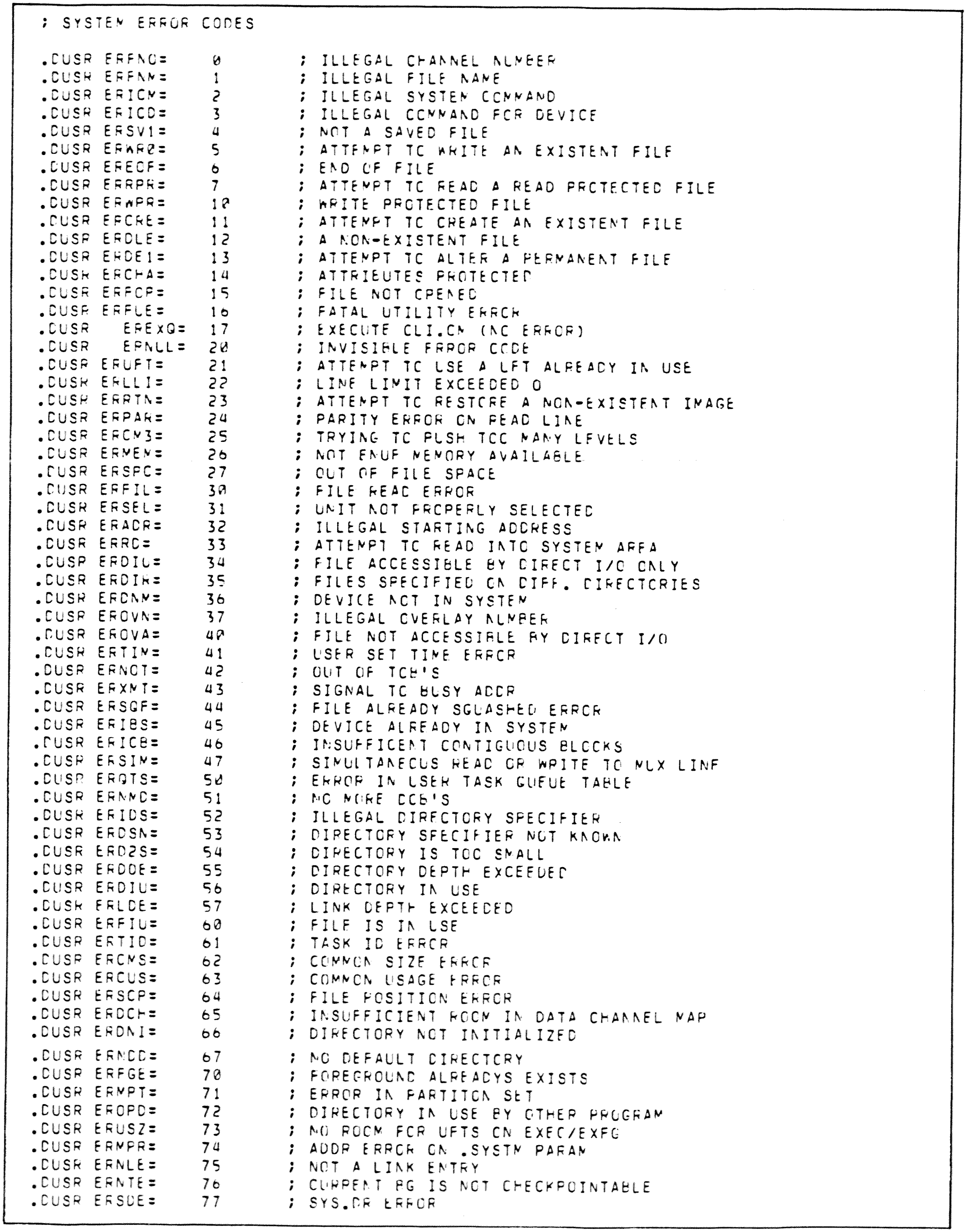
```
; FILE ATTRIBUTES  (IN UFTAT)

.DUSR ATRP =1B0            ;READ PROTECTED
.DUSR ATCHA=1B1            ;CHANGE ATTRIBUTE PROTECTED
.DUSR ATSAV=1B2            ;SAVED FILE
.DUSR ATNRS=1B7            ;CANNOT BE A RESOLUTION ENTRY
.DUSR ATUS1=1B9            ;USER ATTRIBUTE # 1
.DUSR ATUS2=1B10           ;USER ATTRIBUTE # 2
.DUSR ATPER=1B14           ;PERMANENT FILE
.DUSR ATWP =1B15           ;WRITE PROTECTED


; FILE CHARACTERISTICS  (IN UFTAT)

.DUSR ATMSK=7B7            ;TO GET HIGH ORDER PART OF 3330
                           ; ADDRESSES OUT OF UFTDL
.DUSR ATLNK=1B3            ;LINK ENTRY
.DUSR ATPAR=1B4            ;PARTITION ENTRY
.DUSR ATDIR=1B5            ;DIRECTORY ENTRY
.DUSR ATRES=1B6            ;LINK RESOLUTION (TEMPORARY)
.DUSR ATCON=1B12           ;CONTIGUOUS FILE
.DUSR ATRAN=1B13           ;RANDOM FILE

;
; DCT PARAMETERS.
;

.DUSR DCTBS=0              ;1B0=1 => DEVICE USES DATA CHANNEL
.DUSR DCTMS=1              ;MASK OF LOWER PRIORITY DEVICES
.DUSR DCTIS=2              ;ADDRESS OF INTERRUPT SERVICE ROUTINE



; DEVICE CHARACTERISTICS  (IN UFTCH)

.DUSR   DCSTB=  1B15      ; SUPPRESS TRAILING BLANKS $CDR ONLY
.DUSR   DCCPO=  1B15      ; DEVICE REQUIRING LEADER/TRAILER
.DUSR   DCSTO=  1B15      ; USER SPECIFIED TIME OUT CONSTANT (MCA)
.DUSR   DCCGN=  1B14      ; GRAPHICAL OUTPUT DEVICE WITHOUT TABBING
                          ; HARDWARE
.DUSR   DCIDI=  1B13      ; INPUT DEVICE REQUIRING OPERATOR INTERVENTION
.DUSR   DCLCD=  1B12      ; INPUT DEVICE IS 6053-TYPE TERMINAL
.DUSR   DCCNF=  1B12      ; OUTPUT DEVICE WITHOUT FORM FEED HARDWARE
.DUSR   DCTC=   1B11      ; TELETYPE OUTPUT DEVICE
.DUSR   DCKEY=  1B10      ; KEYBOARD DEVICE
.DUSR   DCNAF=  1B09      ; OUTPUT DEVICE REQUIRING NULLS AFTER FORM FEEDS
.DUSR   DCRAT=  1B08      ; RUBOUTS AFTER TABS REQUIRED
.DUSR   DCPCK=  1B07      ; DEVICE REQUIRING PARITY CHECK
.DUSR   DCLAC=  1B06      ; REQUIRES LINE FEEDS AFTER CARRIAGE RTN
.DUSR   DCSPO=  1B05      ; SPOOLABLE DEVICE
.DUSR   DCFWD=  1B04      ; FULL WORD DEVICE (ANYTHING GREATER THAN
.DUSR   DCFFO=  1B03      ; FORM FEEDS ON OPEN

.DUSR   DCLTU=  1B02      ; CHANGE LOWER CASE ASCII TO UPPER
.DUSR   DCC80=  1B01      ; READ 80 COLUMNS
.DUSR   DCDIO=  1B00      ; SUSPEND PROTOCOL ON TRANSMIT (MCA)
.DUSR   DCFDK=  1B00      ; DISK CHARACTERISTIC (SET NON-PARAMETRICALLY)
                          ; SET MEANS ITS 3330
.DUSR   DCSPC=  1B00      ; SPOOL CONTROL
                          ; SET = SPOOLING ENABLED
                          ; RESET = SPOOLING DISABLED
```



```
 ;
 ; DEVICE CHARACTERISTICS FOR QTY AND ALM (PARU.SR)
 ;

  .DUSR   DCNI=     1B15      ;(MASKING ENABLES) CONSOLE INTERRUPTS
 ;.DUSR   DCCGN=    1B14      ;(MASKING DISABLES) TAB EXPANSION
  .DUSR   DCLOC=    1B13      ;LOCAL LINE (MASKING MAKES MODEM LINE)

 ;                  1B12      ;SAVE FOR 3 MODEM PROTOCALS
 ;.DUSR   DCTC=     1B11      ;_ FOR RUBOUT (MASKING GIVES BACKSPACE)
 ;.DUSR   DCKEY=    1B1Ø      ;(MASKING DISABLES) INPUT ECHOING,
                              ;        LINE EDITS, AND ↑Z EOF

 ;.DUSR   DCNAF=    1B9       ;(MASKING DISABLES) 2Ø NULLS AFTER FORM FEED
  .DUSR   DCXON=    1B8       ;(MASKING ENABLES) XON/XOFF PROTOCALL FOR $TTR
 ;                  1B7       ;SAVE FOR FUTURE USE

 ;.DUSR   DCLAC=    1B6       ;(MASKING DISABLES) LINE FEED AFTER CARRAIGE RETURN
 ;.DUSR   DCSFO=    1B5       ;(MUST BE OFF) SPOOLING
  .DUSR   DCCRE=    1B4       ;CARRAIGE RETURN ECHO
                              ;        (MASKING ENABLES CR AS ENTER KEY)
 ;
 ; .WRL TO LINE 64
 ;
 ;        ACØ= CODE+LINE #
 ;        AC1= DATA

 .DUSR    W64DC=  ØB7             ;CHANGE DEVICE CHARACTERISTIC MASK (AC1)
 .DUSR    W64LS=  1B7             ;CHANGE LINE SPEED (AC1= Ø -> 3)
 .DUSR    W64MS=  2B7             ;CHANGE MODEM STATE (AC1) AS FOLLOWS
 .DUSR      W64DTR= 1B15          ;  RAISE DATA TERMINAL READY
                                  ;       ELSE LOWER
 .DUSR      W64RTS= 1B14          ;  RAISE REQUEST TO SEND
                                  ;       ELSE LOWER


 ;
 ; SWITCHES
 ;

 .DUSR    A.SW=    1BØØ
 .DUSR    B.SW=    1BØ1
 .DUSR    C.SW=    1BØ2
 .DUSR    D.SW=    1BØ3
 .DUSR    E.SW=    1BØ4
 .DUSR    F.SW=    1BØ5
 .DUSR    G.SW=    1BØ6
 .DUSR    H.SW=    1BØ7
 .DUSR    I.SW=    1BØ8
 .DUSR    J.SW=    1BØ9
 .DUSR    K.SW=    1B1Ø
 .DUSR    L.SW=    1B11
 .DUSR    M.SW=    1B12
 .DUSR    N.SW=    1B13
 .DUSR    O.SW=    1B14
 .DUSR    P.SW=    1B15
 .DUSR    Q.SW=    1BØØ
 .DUSR    R.SW=    1BØ1
 .DUSR    S.SW=    1BØ2
 .DUSR    T.SW=    1BØ3
 .DUSR    U.SW=    1BØ4
 .DUSR    V.SW=    1BØ5
 .DUSR    W.SW=    1BØ6
 .DUSR    X.SW=    1BØ7
 .DUSR    Y.SW=    1BØ8
 .DUSR    Z.SW=    1BØ9
```

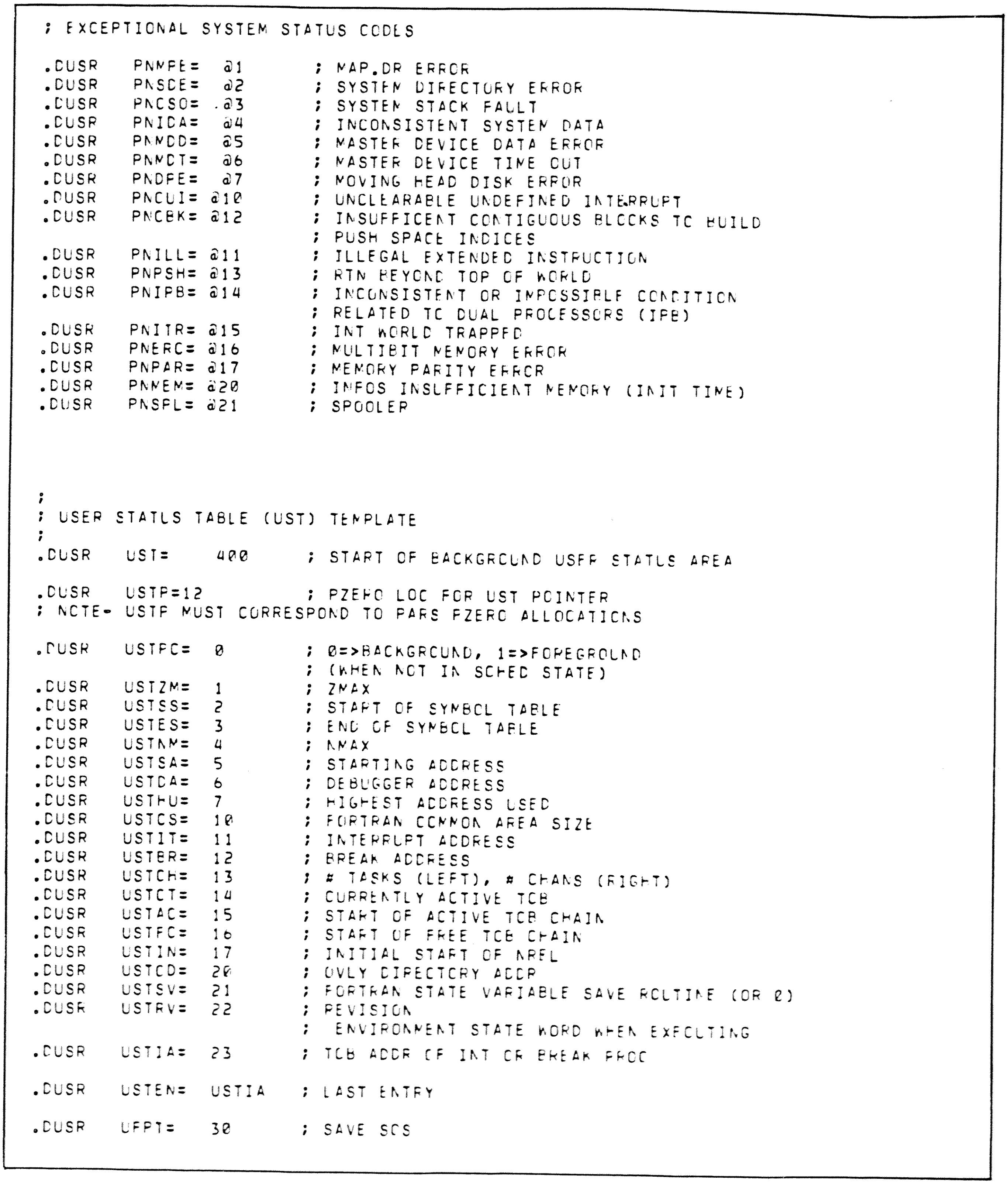


```
;
; SYSTEM CONSTANTS
;

.DUSR SCWPB=255.          ;WORDS PER BLOCK
.DUSR SCDBS=256.          ;SIZE OF DISK BLOCK
.DUSR SCRRL=64.           ;WORDS PER RANDOM RECORD
.DUSR SCLLG=132.          ;MAX LINE LENGTH
.DUSR SCAMX=24.           ;MAX ARGUMENT LENGTH IN BYTES
.DUSR SCFNL=UFTEX-UFTFN+1 ;FILE NAME LENGTH
.DUSR SCEXT=UFTEX-UFTFN   ;EXTENSION OFFSET IN NAME AREA
.DUSR SCMER=10.           ;MAX ERROR RETRY COUNT
.DUSR SCSTR=16            ;SAVE FILE STARTING ADDRESS
.DUSR SCTIM=-80.          ;RINGIO 1 MS. LOOP TIME (SN)
.DUSR SCPPL=0             ;PRIMARY PARTITION LEVEL
.DUSR SCPPA=6             ;PRIMARY PARTITION BASE ADDRESS
.DUSR SCDSK=3             ;ABSOLUTE ADDRESS OF DISK INFORMATION BLOCK
.DUSR SCBAD=4             ;ABSOLUTE ADDRESS OF BAD BLOCK TABLE BLOCK
.DUSR SCSYS=0             ;SYS.DR ADDRESS OFFSET
.DUSR SCPSH=1             ;PUSH DIRECTORY OFFSET
.DUSR SCPNM=4             ;MAX NUMBER OF PUSH LEVELS
.DUSR SCMAP=SCPNM*2+SCPSH ;RELATIVE BASE ADDRESS OF MAP.DR
.DUSR SCBPB=1             ;RELATIVE BACKGROUND PUSH BASE
.DUSR SCFPB=SCBPB+SCPNM   ;RELATIVE FOREGROUND PUSH BASE
.DUSR SCFZW=SCPNM*4+SCBPB ;FRAME SIZE WORD (SKIP DOUBLE WORD PUSH INDICES)
.DUSR SCNVW=SCFZW+1       ;NUMBER-OF-SYSTEM-OVERLAYS WORD
.DUSR SFINT=1B0           ;INTERRUPT FLAG
.DUSR SFBRK=1B15          ;BREAK FLAG
.DUSR SCNSC=64.           ;NUMBER OF SYSTEM OVERLAYS


; SYSTEM BOOTSTRAP CONSTANTS

.DUSR   SCTBP=0          ;TEXT STRING BYTE POINTER
.DUSR   SCINS=1          ;SWITCHED FULL/PARTIAL-OVERLAYS ADDRESS
.DUSR   SCPSA=2          ;PROGRAM START ADDRESS
.DUSR   SCPAR=SCPSA      ;PARTIAL INIT ADDRESS
.DUSR   SCINT=3          ;FULL/PARTIAL-OVERLAYS INIT ADDRESS
.DUSR   SCCLI=SCINT+1    ;ADDRESS OF END OF CLI
.DUSR   SCZMX=SCCLI+1    ;SQUASHED/UNSQUASHED FLAG
.DUSR   SCCPL=SCZMX+1    ;CURRENT PARTITION LEVEL
.DUSR   SCPBA=SCCPL+1    ;PARTITION BASE ADDRESS (LOW ORDER)
.DUSR   SCOFA=SCPBA+1    ;OVERLAY BASE ADDRESS (LOW ORDER)
.DUSR   SCPB1=SCOFA+1    ;PARTITION BASE ADDRESS (HIGH ORDER)
.DUSR   SCOF1=SCPB1+1    ;OVERLAY BASE ADDRESS (HIGH ORDER)
.DUSR   SCBAS=SCOF1+1    ;BASE OF INFORMATION BLOCK
.DUSR   SCSWC=SCBAS      ;SWITCH FOR SCINS ENTRY
.DUSR   SCIDV=20         ;INITIAL DEVICE CODE
.DUSR   SCALN=0          ;ASCII UNIT NUMBER
.DUSR   SCUN=1           ;UNIT (DEVICE CODE)
.DUSR   SCGC=2           ;ENTRY TO PASS FILENAME
.DUSR   SCNGO=4          ;ENTRY TO ASK FROM CONSOLE
```

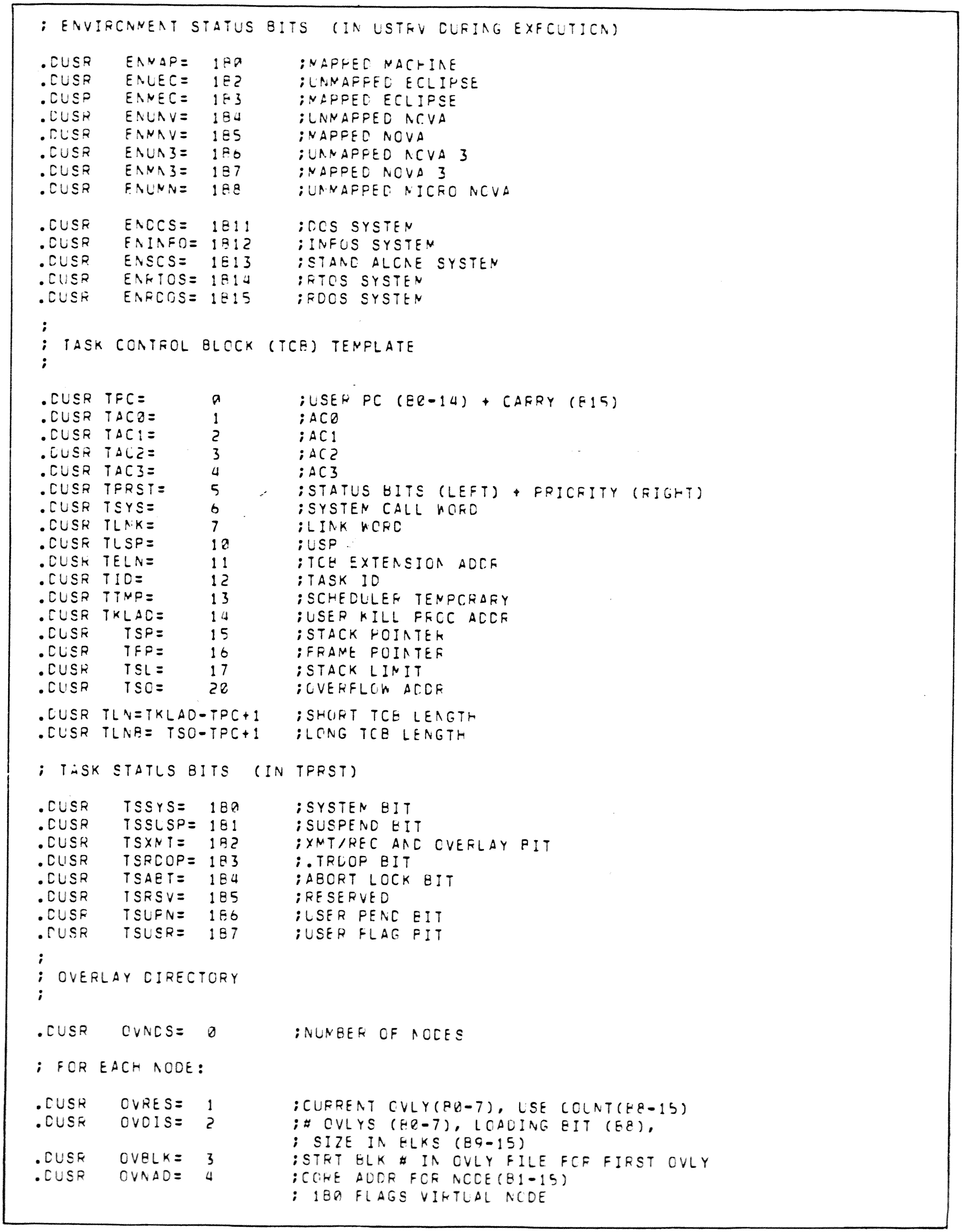


```
; SYSTEM ERROR CODES

.DUSR ERFNO=     0         ; ILLEGAL CHANNEL NUMBER
.DUSR ERFNM=     1         ; ILLEGAL FILE NAME
.DUSR ERICM=     2         ; ILLEGAL SYSTEM COMMAND
.DUSR ERICD=     3         ; ILLEGAL COMMAND FOR DEVICE
.DUSR ERSV1=     4         ; NOT A SAVED FILE
.DUSR ERWRO=     5         ; ATTEMPT TO WRITE AN EXISTENT FILE
.DUSR EREOF=     6         ; END OF FILE
.DUSR ERRPR=     7         ; ATTEMPT TO READ A READ PROTECTED FILE
.DUSR ERWPR=     10        ; WRITE PROTECTED FILE
.DUSR ERCRE=     11        ; ATTEMPT TO CREATE AN EXISTENT FILE
.DUSR ERDLE=     12        ; A NON-EXISTENT FILE
.DUSR ERDE1=     13        ; ATTEMPT TO ALTER A PERMANENT FILE
.DUSR ERCHA=     14        ; ATTRIBUTES PROTECTED
.DUSR ERFOP=     15        ; FILE NOT OPENED
.DUSR ERFUE=     16        ; FATAL UTILITY ERROR
.DUSR    EREXQ=  17        ; EXECUTE CLI.CM (NO ERROR)
.DUSR    ERNUL=  20        ; INVISIBLE ERROR CODE
.DUSR ERUFT=     21        ; ATTEMPT TO USE A UFT ALREADY IN USE
.DUSR ERLLI=     22        ; LINE LIMIT EXCEEDED O
.DUSR ERRTN=     23        ; ATTEMPT TO RESTORE A NON-EXISTENT IMAGE
.DUSR ERPAR=     24        ; PARITY ERROR ON READ LINE
.DUSR ERCM3=     25        ; TRYING TO PUSH TOO MANY LEVELS
.DUSR ERMEM=     26        ; NOT ENUF MEMORY AVAILABLE
.DUSR ERSPC=     27        ; OUT OF FILE SPACE
.DUSR ERFIL=     30        ; FILE READ ERROR
.DUSR ERSEL=     31        ; UNIT NOT PROPERLY SELECTED
.DUSR ERADR=     32        ; ILLEGAL STARTING ADDRESS
.DUSR ERRD=      33        ; ATTEMPT TO READ INTO SYSTEM AREA
.DUSR ERDIO=     34        ; FILE ACCESSIBLE BY DIRECT I/O ONLY
.DUSR ERDIR=     35        ; FILES SPECIFIED ON DIFF. DIRECTORIES
.DUSR ERDNM=     36        ; DEVICE NOT IN SYSTEM
.DUSR EROVN=     37        ; ILLEGAL OVERLAY NUMBER
.DUSR EROVA=     40        ; FILE NOT ACCESSIBLE BY DIRECT I/O
.DUSR ERTIM=     41        ; USER SET TIME ERROR
.DUSR ERNOT=     42        ; OUT OF TCB'S
.DUSR ERXMT=     43        ; SIGNAL TO BUSY ADDR
.DUSR ERSQF=     44        ; FILE ALREADY SQUASHED ERROR
.DUSR ERIBS=     45        ; DEVICE ALREADY IN SYSTEM
.DUSR ERICB=     46        ; INSUFFICENT CONTIGUOUS BLOCKS
.DUSR ERSIM=     47        ; SIMULTANEOUS READ OR WRITE TO MUX LINE
.DUSR ERQTS=     50        ; ERROR IN USER TASK QUEUE TABLE
.DUSR ERNMC=     51        ; NO MORE CCB'S
.DUSR ERIDS=     52        ; ILLEGAL DIRECTORY SPECIFIER
.DUSR ERDSN=     53        ; DIRECTORY SPECIFIER NOT KNOWN
.DUSR ERD2S=     54        ; DIRECTORY IS TOO SMALL
.DUSR ERDDE=     55        ; DIRECTORY DEPTH EXCEEDED
.DUSR ERDIU=     56        ; DIRECTORY IN USE
.DUSR ERLDE=     57        ; LINK DEPTH EXCEEDED
.DUSR ERFIU=     60        ; FILE IS IN USE
.DUSR ERTID=     61        ; TASK ID ERROR
.DUSR ERCMS=     62        ; COMMON SIZE ERROR
.DUSR ERCUS=     63        ; COMMON USAGE ERROR
.DUSR ERSCP=     64        ; FILE POSITION ERROR
.DUSR ERDCH=     65        ; INSUFFICIENT ROOM IN DATA CHANNEL MAP
.DUSR ERDNI=     66        ; DIRECTORY NOT INITIALIZED
.DUSR ERNDD=     67        ; NO DEFAULT DIRECTORY
.DUSR ERFGE=     70        ; FOREGROUND ALREADYS EXISTS
.DUSR ERMPT=     71        ; ERROR IN PARTITON SET
.DUSR EROPD=     72        ; DIRECTORY IN USE BY OTHER PROGRAM
.DUSR ERUSZ=     73        ; NO ROOM FOR UFTS ON EXEC/EXFG
.DUSR ERMPR=     74        ; ADDR ERROR ON .SYSTM PARAM
.DUSR ERNLE=     75        ; NOT A LINK ENTRY
.DUSR ERNTE=     76        ; CURRENT BG IS NOT CHECKPOINTABLE
.DUSR ERSDE=     77        ; SYS.DR ERROR
```

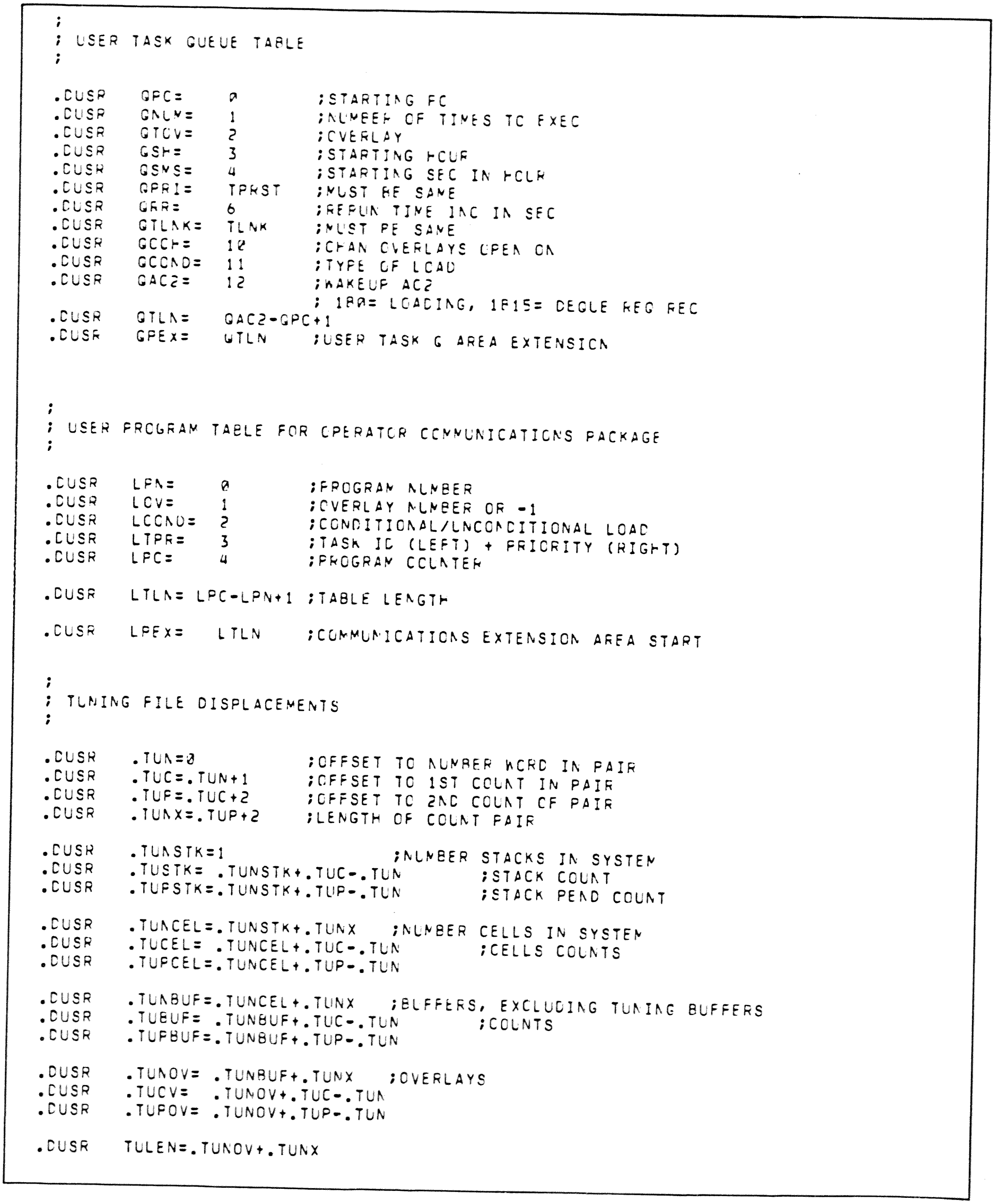
```
.DUSR ERMDE=    100      ; MAP.DF ERROR
.DUSR ERDTO=    101      ; DEVICE TIME OUT
.DUSR ERENA=    102      ; ENTRY NOT ACCESSIBLE VIA LINK
.DUSR ERMCA=    103      ; MCA REQUEST OUTSTANDING
.DUSR ERSRR=    104      ; INCOMPLETE TRANSMISSION CAUSED BY RECIEVER
.DUSR ERSDL=    105      ; SYSTEM DEADLOCK
.DUSR ERCLO=    106      ; I/O TERMINATED BY CHANNEL CLOSE
.DUSR ERSFA=    107      ; SPOOL FILE(S) ACTIVE
.DUSR ERABT=    110      ; TASK NOT FOUND FOR ABORT
.DUSR ERDOP=    111      ; DEVICE PREVIOUSLY OPENED
.DUSR EROVF=    112      ; SYSTEM STACK OVERFLOW
.DUSR ERNMC=    113      ; NO MCA RECEIVE REQUEST OUTSTANDING
.DUSR ERNIR=    114      ; NO INIT/RELEASE ON OPENED DEVICE (MAG TAPE)
.DUSR ERXMZ=    115      ; .XMT & .IXMT MESSAGES MUST BE NON-ZERO
.DUSR ERCANT=   116      ; 'YOU CAN'T DO THAT'
.DUSR ERQOV=    117      ; .TOVLD NOT LOADED FOR QUEUED OVERLAY TASKS
.DUSR EROPM=    120      ; OPERATOR MESSAGE MODULE NOT SYSGENED
.DUSR ERFMT=    121      ; DISK FORMAT ERROR
.DUSR ERBAD=    122      ; DISK HAS INVALID BAD BLOCK TABLE
.DUSR ERBSPC=   123      ; INSUFFICIENT SPACE IN BAD BLOCK POOL (CORE)
.DUSR ERZCB=    124      ; ATTEMPT TO CREATE CONTIG OF ZERO LENGTH
.DUSR ERNSE=    125      ; PROGRAM IS NOT SWAPPABLE
.DUSR ERBLT=    126      ; BLANK TAPE
.DUSR ERRDY=    127      ; LINE NOT READY
.DUSR ERINT=    130      ; CONSOLE INTERRUPT RECEIVED
.DUSR EROVR=    131      ; CHARACTER OVER RUN ERROR
.DUSR ERFRM=    132      ; CHARACTER FRAMING ERROR
.DUSR ERSPT=    133      ; TOO MANY SOFT ERRORS (DOS ONLY)



; CLI ERROR CODES

.DUSR    CNEAR=  300     ; NOT ENOUGH ARGUMENTS
.DUSR    CILAT=  301     ; ILLEGAL ATTRIBUTE
.DUSR    CNDBD=  302     ; NO DEBUG ADDRESS
.DUSR    CCLTL=  303     ; COMMAND LINE TOO LONG
.DUSR    CNSAD=  304     ; NO STARTING ADDRESS
.DUSR    CCKER=  305     ; CHECKSUM ERROR
.DUSR    CNSFS=  306     ; NO SOURCE FILE SPECIFIED
.DUSR    CNACM=  307     ; NOT A COMMAND
.DUSR    CILBK=  310     ; ILLEGAL BLOCK TYPE
.DUSR    CSPER=  311     ; NO FILES MATCH SPECIFIER
.DUSR    CPHER=  312     ; PHASE ERROR
.DUSR    CTMAR=  313     ; TOO MANY ARGUMENTS
.DUSR    CTMAD=  314     ; TOO MANY ACTIVE DEVICES
.DUSR    CILNA=  315     ; ILLEGAL NUMERIC ARGUMENT
.DUSR    CSFLE=  316     ; FATAL SYSTEM UTILITY ERROR
.DUSR    CILAR=  317     ; ILLEGAL ARGUMENT
.DUSR    CCANT=  320     ; IMPROPER OR MALICIOUS INPUT
.DUSR    CTMLI=  321     ; TOO MANY LEVELS OF INDIRECT FILES
.DUSR    CSYER=  322     ; SYNTAX ERROR
.DUSR    CBKER=  323     ; BRACKET ERROR
.DUSR    CPARE=  324     ; PAREN ERROR
.DUSR    CCART=  325     ; < WITHOUT > OR > WITHOUT <
.DUSR    CCAR1=  326     ; ILLEGAL NESTING OF <> AND ()
.DUSR    CINDE=  327     ; ILLEGAL INDIRECT FILENAME
.DUSR    CPAR1=  330     ; ILLEGAL NESTING OF () AND []
.DUSR    CIVAR=  331     ; ILLEGAL VARIABLE
.DUSR    CILTA=  332     ; ILLEGAL TEXT ARGUMENT
.DUSR    CTATL=  333     ; TEXT ARGUMENT TOO LONG

.DUSR    CCMAX=  CTATL   ; MAX CLI ERROR CODE
.DUSR    ERML=   30.     ; MAXIMUM ERROR MESSAGE LENGTH
```



```
; EXCEPTIONAL SYSTEM STATUS CODES

.DUSR   PNMPE=  @1        ; MAP.DR ERROR
.DUSR   PNSDE=  @2        ; SYSTEM DIRECTORY ERROR
.DUSR   PNCSO= .@3        ; SYSTEM STACK FAULT
.DUSR   PNICA=  @4        ; INCONSISTENT SYSTEM DATA
.DUSR   PNMDD=  @5        ; MASTER DEVICE DATA ERROR
.DUSR   PNMDT=  @6        ; MASTER DEVICE TIME OUT
.DUSR   PNDPE=  @7        ; MOVING HEAD DISK ERROR
.DUSR   PNCUI= @10        ; UNCLEARABLE UNDEFINED INTERRUPT
.DUSR   PNCBK= @12        ; INSUFFICENT CONTIGUOUS BLOCKS TO BUILD
                          ; PUSH SPACE INDICES
.DUSR   PNILL= @11        ; ILLEGAL EXTENDED INSTRUCTION
.DUSR   PNPSH= @13        ; RTN BEYOND TOP OF WORLD
.DUSR   PNIPB= @14        ; INCONSISTENT OR IMPOSSIBLE CONDITION
                          ; RELATED TO DUAL PROCESSORS (IPB)
.DUSR   PNITR= @15        ; INT WORLD TRAPPED
.DUSR   PNERC= @16        ; MULTIBIT MEMORY ERROR
.DUSR   PNPAR= @17        ; MEMORY PARITY ERROR
.DUSR   PNMEM= @20        ; INFOS INSUFFICIENT MEMORY (INIT TIME)
.DUSR   PNSPL= @21        ; SPOOLER



;
; USER STATUS TABLE (UST) TEMPLATE
;
.DUSR   UST=     400      ; START OF BACKGROUND USER STATUS AREA

.DUSR   USTP=12           ; PZERO LOC FOR UST POINTER
; NOTE- USTP MUST CORRESPOND TO PARS PZERO ALLOCATIONS

.DUSR   USTPC=  0         ; 0=>BACKGROUND, 1=>FOREGROUND
                          ; (WHEN NOT IN SCHED STATE)
.DUSR   USTZM=  1         ; ZMAX
.DUSR   USTSS=  2         ; START OF SYMBOL TABLE
.DUSR   USTES=  3         ; END OF SYMBOL TABLE
.DUSR   USTNM=  4         ; NMAX
.DUSR   USTSA=  5         ; STARTING ADDRESS
.DUSR   USTDA=  6         ; DEBUGGER ADDRESS
.DUSR   USTHU=  7         ; HIGHEST ADDRESS USED
.DUSR   USTCS=  10        ; FORTRAN COMMON AREA SIZE
.DUSR   USTIT=  11        ; INTERRUPT ADDRESS
.DUSR   USTBR=  12        ; BREAK ADDRESS
.DUSR   USTCH=  13        ; # TASKS (LEFT), # CHANS (RIGHT)
.DUSR   USTCT=  14        ; CURRENTLY ACTIVE TCB
.DUSR   USTAC=  15        ; START OF ACTIVE TCB CHAIN
.DUSR   USTFC=  16        ; START OF FREE TCB CHAIN
.DUSR   USTIN=  17        ; INITIAL START OF NREL
.DUSR   USTOD=  20        ; OVLY DIRECTORY ADDR
.DUSR   USTSV=  21        ; FORTRAN STATE VARIABLE SAVE ROUTINE (OR 0)
.DUSR   USTRV=  22        ; REVISION
                          ;  ENVIRONMENT STATE WORD WHEN EXECUTING
.DUSR   USTIA=  23        ; TCB ADDR OF INT OR BREAK PROC

.DUSR   USTEN=  USTIA     ; LAST ENTRY

.DUSR   UFPT=   30        ; SAVE SCS
```



```
; ENVIRONMENT STATUS BITS  (IN USTRV DURING EXECUTION)

.DUSR    ENMAP=  1BØ      ;MAPPED MACHINE
.DUSR    ENUEC=  1B2      ;UNMAPPED ECLIPSE
.DUSR    ENMEC=  1B3      ;MAPPED ECLIPSE
.DUSR    ENUNV=  1B4      ;UNMAPPED NOVA
.DUSR    ENMNV=  1B5      ;MAPPED NOVA
.DUSR    ENUN3=  1B6      ;UNMAPPED NOVA 3
.DUSR    ENMN3=  1B7      ;MAPPED NOVA 3
.DUSR    ENUMN=  1B8      ;UNMAPPED MICRO NOVA

.DUSR    ENDOS=  1B11     ;DOS SYSTEM
.DUSR    ENINFO= 1B12     ;INFOS SYSTEM
.DUSR    ENSOS=  1B13     ;STAND ALONE SYSTEM
.DUSR    ENRTOS= 1B14     ;RTOS SYSTEM
.DUSR    ENRDOS= 1B15     ;RDOS SYSTEM
;
; TASK CONTROL BLOCK (TCB) TEMPLATE
;

.DUSR TPC=       Ø        ;USER PC (BØ-14) + CARRY (B15)
.DUSR TACØ=      1        ;ACØ
.DUSR TAC1=      2        ;AC1
.DUSR TAC2=      3        ;AC2
.DUSR TAC3=      4        ;AC3
.DUSR TPRST=     5        ;STATUS BITS (LEFT) + PRIORITY (RIGHT)
.DUSR TSYS=      6        ;SYSTEM CALL WORD
.DUSR TLNK=      7        ;LINK WORD
.DUSR TUSP=      1Ø       ;USP
.DUSR TELN=      11       ;TCB EXTENSION ADDR
.DUSR TID=       12       ;TASK ID
.DUSR TTMP=      13       ;SCHEDULER TEMPORARY
.DUSR TKLAD=     14       ;USER KILL PROC ADDR
.DUSR   TSP=     15       ;STACK POINTER
.DUSR   TFP=     16       ;FRAME POINTER
.DUSR   TSL=     17       ;STACK LIMIT
.DUSR   TSO=     2Ø       ;OVERFLOW ADDR

.DUSR TLN=TKLAD-TPC+1     ;SHORT TCB LENGTH
.DUSR TLNB= TSO-TPC+1     ;LONG TCB LENGTH

; TASK STATUS BITS  (IN TPRST)

.DUSR    TSSYS=  1BØ      ;SYSTEM BIT
.DUSR    TSSUSP= 1B1      ;SUSPEND BIT
.DUSR    TSXMT=  1B2      ;XMT/REC AND OVERLAY BIT
.DUSR    TSRDOP= 1B3      ;.TRDOP BIT
.DUSR    TSABT=  1B4      ;ABORT LOCK BIT
.DUSR    TSRSV=  1B5      ;RESERVED
.DUSR    TSUPN=  1B6      ;USER PEND BIT
.DUSR    TSUSR=  1B7      ;USER FLAG BIT
;
; OVERLAY DIRECTORY
;

.DUSR    OVNDS=  Ø        ;NUMBER OF NODES

; FOR EACH NODE:

.DUSR    OVRES=  1        ;CURRENT OVLY(BØ-7), USE COUNT(B8-15)
.DUSR    OVDIS=  2        ;# OVLYS (BØ-7), LOADING BIT (B8),
                          ; SIZE IN BLKS (B9-15)
.DUSR    OVBLK=  3        ;STRT BLK # IN OVLY FILE FOR FIRST OVLY
.DUSR    OVNAD=  4        ;CORE ADDR FOR NODE(B1-15)
                          ; 1BØ FLAGS VIRTUAL NODE
```



```
;
; USER TASK QUEUE TABLE
;

.DUSR   QPC=     0          ;STARTING PC
.DUSR   QNUM=    1          ;NUMBER OF TIMES TO EXEC
.DUSR   QTOV=    2          ;OVERLAY
.DUSR   QSH=     3          ;STARTING HOUR
.DUSR   QSMS=    4          ;STARTING SEC IN HOUR
.DUSR   QPRI=    TPRST      ;MUST BE SAME
.DUSR   QRR=     6          ;RERUN TIME INC IN SEC
.DUSR   QTLNK=   TLNK       ;MUST BE SAME
.DUSR   QCCH=    10         ;CHAN OVERLAYS OPEN ON
.DUSR   QCOND=   11         ;TYPE OF LOAD
.DUSR   QAC2=    12         ;WAKEUP AC2
                            ; 1B0= LOADING, 1B15= DOUBLE REQ REC
.DUSR   QTLN=    QAC2-QPC+1
.DUSR   QPEX=    QTLN       ;USER TASK Q AREA EXTENSION



;
; USER PROGRAM TABLE FOR OPERATOR COMMUNICATIONS PACKAGE
;

.DUSR   LPN=     0          ;PROGRAM NUMBER
.DUSR   LOV=     1          ;OVERLAY NUMBER OR -1
.DUSR   LCOND=   2          ;CONDITIONAL/UNCONDITIONAL LOAD
.DUSR   LTPR=    3          ;TASK ID (LEFT) + PRIORITY (RIGHT)
.DUSR   LPC=     4          ;PROGRAM COUNTER

.DUSR   LTLN= LPC-LPN+1 ;TABLE LENGTH

.DUSR   LPEX=    LTLN       ;COMMUNICATIONS EXTENSION AREA START


;
; TUNING FILE DISPLACEMENTS
;

.DUSR   .TUN=0              ;OFFSET TO NUMBER WORD IN PAIR
.DUSR   .TUC=.TUN+1         ;OFFSET TO 1ST COUNT IN PAIR
.DUSR   .TUP=.TUC+2         ;OFFSET TO 2ND COUNT OF PAIR
.DUSR   .TUNX=.TUP+2        ;LENGTH OF COUNT PAIR

.DUSR   .TUNSTK=1                       ;NUMBER STACKS IN SYSTEM
.DUSR   .TUSTK= .TUNSTK+.TUC-.TUN               ;STACK COUNT
.DUSR   .TUPSTK=.TUNSTK+.TUP-.TUN               ;STACK PEND COUNT

.DUSR   .TUNCEL=.TUNSTK+.TUNX   ;NUMBER CELLS IN SYSTEM
.DUSR   .TUCEL= .TUNCEL+.TUC-.TUN               ;CELLS COUNTS
.DUSR   .TUPCEL=.TUNCEL+.TUP-.TUN

.DUSR   .TUNBUF=.TUNCEL+.TUNX   ;BUFFERS, EXCLUDING TUNING BUFFERS
.DUSR   .TUBUF= .TUNBUF+.TUC-.TUN               ;COUNTS
.DUSR   .TUPBUF=.TUNBUF+.TUP-.TUN

.DUSR   .TUNOV= .TUNBUF+.TUNX   ;OVERLAYS
.DUSR   .TUOV=  .TUNOV+.TUC-.TUN
.DUSR   .TUPOV= .TUNOV+.TUP-.TUN

.DUSR   TULEN=.TUNOV+.TUNX
```

End of Appendix

Appendix C
Interrevision Changes

This appendix outlines some major changes in RDOS between revisions 5.00 and 6.00. Perhaps the most significant change involves the system library, SYS.LB. This has global importance because RLDR uses code from SYS.LB to build every RDOS save file.

Changes often occur in RDOS within revisions (e.g., between revisions 6.10 and 6.20); also, patch updates are often issued after a revision is released. Thus you should consult the Release Notice and Update Notice supplied with the RDOS software for specific details.

SYSGEN

For RDOS revision 6.00, the disk initializer, DKINIT.SV, BOOT.SV and the starter system (BOOTSYS.SV or FBOOTSYS.SV) all support several new disks and devices. The starter system no longer supports a line printer; to use a printer, you must generate a tailored system.

Revision 6.00 and later SYSGENs cannot use pre-revision 6.00 dialog files (SYS.SG or equivalent). In fact, revision 6.4 SYSGEN cannot use pre-revision 6.4 dialog files. (Every time a new question is added to SYSGEN, the SYSGEN with the new question becomes incompatible with older dialog files.)

Other

The revision 6.00 CLI supports some new features (e.g., macro files); thus revision 6.00 and later CLIs may not run with pre-revision 6.00 RDOS systems.

Certain revision 6.00 utility programs (e.g., MAC) may not run under a pre-revision 6.00 RDOS system.

The system library (SYS.LB) now provides more support for high-level Data General languages. The new interface within this library is called the Universal MultiTasking Interface (UMTI), and future high-level language development for RDOS will be based on it. Programs built with a pre-revision 6.00 SYS.LB may not execute under a revision 6.00 (or later) RDOS system; we also recommend that you reload your old programs using the new SYS.LB, if possible. Some older compilers, however, produce code incompatible with the new SYS.LB. To allow you to use these, we have provided an updated version of the old system library; its name is SYS5.LB. You can use this library to load programs produced by your old compiler if you choose not to replace your old compilers with new ones.

Another change for revision 6.00 is that we no longer supply the Task Monitor sources with the RDOS software; if you want them, you must order them specifically. Because of this, we have described some major features of the new monitor in Appendix J.

End of Appendix

Appendix D
Real-Time Programming Examples

This appendix contains two examples of assembly-language programs written for a real-time environment.

TIMEC Program

The first example is TIMEC, a bare-bones program which creates an additional task (TASK) at the same priority. (See Figures D-2 and D-3.)

During execution, TIMEC creates at the same priority as itself (0). Task competes for CPU control, gets it when TIMEC suspends itself, and retains it until *it* suspends itself. When each task gains control, it prints a message on the console. TIMEC suspends itself for 2 seconds, and TASK suspends itself for 4 seconds. After about eight seconds, the console shows:

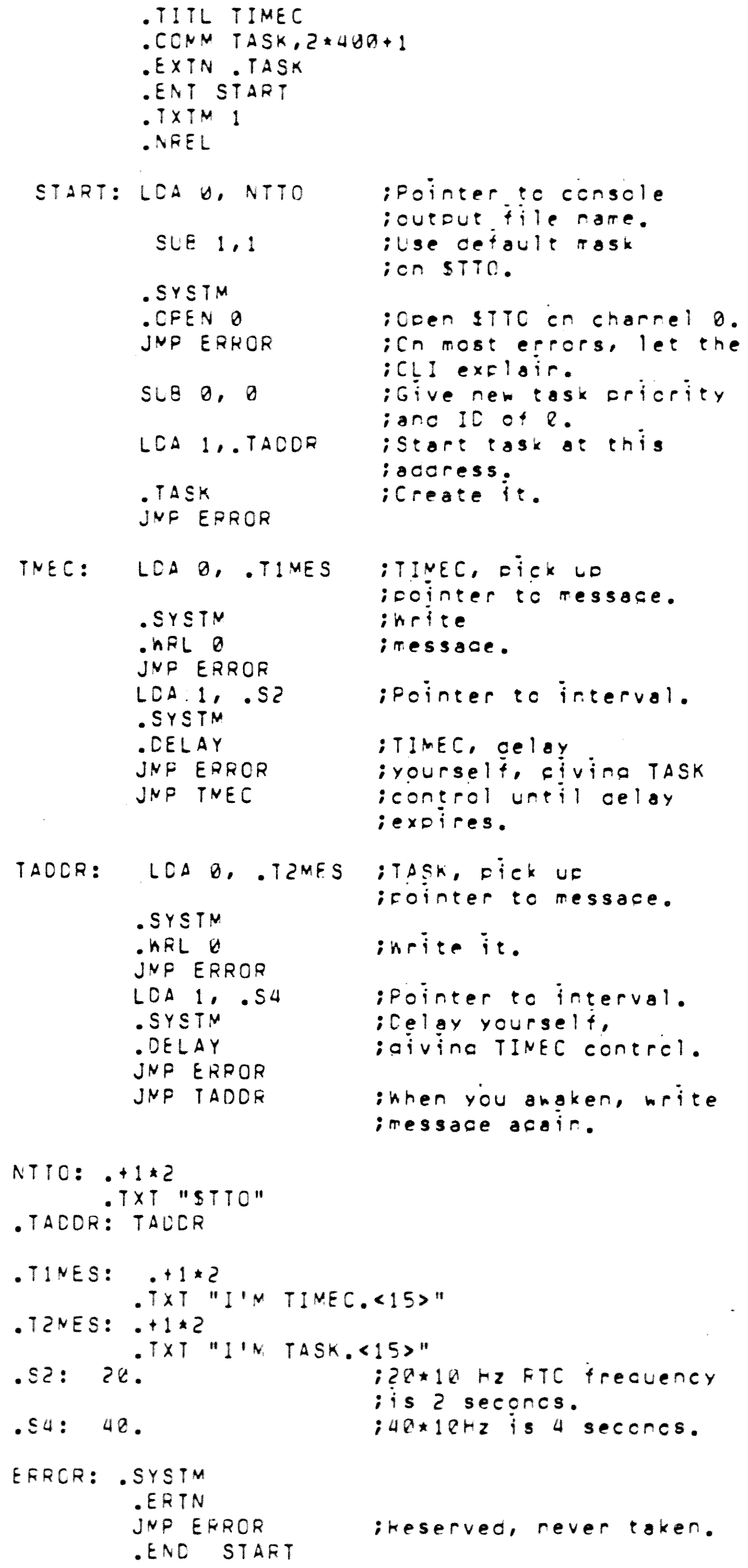
```
I'M TIMEC
I'M TASK
I'M TIMEC
I'M TASK
I'M TIMEC
I'M TIMEC
I'M TASK
I'M TIMEC
```

The messages appear in syncopated fashion because the tasks suspend themselves for different times as shown in Figure D-1.

TIMEC includes no code to return to the CLI, therefore you must use an RDOS interrupt (CTRL-A or CTRL-C) to stop it and return to the CLI.

```
         I'M TIMEC  I'M TIMEC  I'M TASK   I'M TIMEC  I'M TASK
seconds      0          2          4          6          8
         I'M TASK              I'M TIMEC             I'M TIMEC
```

Figure D-1. TIMEC and TASK Messages



```
          .TITL TIMEC
          .COMM TASK,2*400+1
          .EXTN .TASK
          .ENT START
          .TXTM 1
          .NREL

 START:   LDA 0, NTTO        ;Pointer to console
                             ;output file name.
           SUB 1,1           ;Use default mask
                             ;on $TTO.
          .SYSTM
          .OPEN 0            ;Open $TTO on channel 0.
          JMP ERROR          ;On most errors, let the
                             ;CLI explain.
          SUB 0, 0           ;Give new task priority
                             ;and ID of 0.
          LDA 1,.TADDR       ;Start task at this
                             ;address.
          .TASK              ;Create it.
          JMP ERROR

TMEC:     LDA 0, .T1MES      ;TIMEC, pick up
                             ;pointer to message.
          .SYSTM             ;Write
          .WRL 0             ;message.
          JMP ERROR
          LDA 1, .S2         ;Pointer to interval.
          .SYSTM
          .DELAY             ;TIMEC, delay
          JMP ERROR          ;yourself, giving TASK
          JMP TMEC           ;control until delay
                             ;expires.

TADDR:     LDA 0, .T2MES     ;TASK, pick up
                             ;pointer to message.
          .SYSTM
          .WRL 0             ;Write it.
          JMP ERROR
          LDA 1, .S4         ;Pointer to interval.
          .SYSTM             ;Delay yourself,
          .DELAY             ;giving TIMEC control.
          JMP ERROR
          JMP TADDR          ;When you awaken, write
                             ;message again.

NTTO: .+1*2
      .TXT "$TTO"
.TADDR: TADDR

.T1MES:   .+1*2
          .TXT "I'M TIMEC.<15>"
.T2MES: .+1*2
          .TXT "I'M TASK.<15>"
.S2:  20.                    ;20*10 Hz RTC frequency
                             ;is 2 seconds.
.S4:  40.                    ;40*10Hz is 4 seconds.

ERROR: .SYSTM
          .ERTN
          JMP ERROR          ;Reserved, never taken.
          .END  START
```

Figure D-2. TIMEC Program Listing

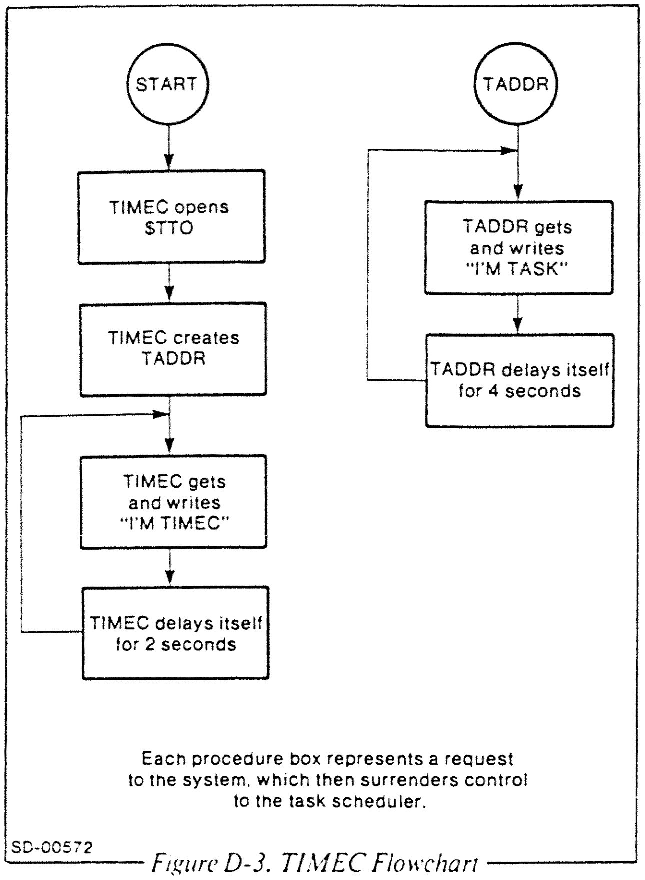


Figure D-3. TIMEC Flowchart

Example Program

The second program, EXAMPLE, is a multitasking program that uses overlays; it shows multitask overlay calls and a queued overlay task. The assembler listing for EXAMPLE, and its two overlays, QUE and COMP, appear in Figure D-4; a flowchart follows in Figure D-5.

In EXAMPLE, the main program task opens the console input, output, and overlay files; then it sets its priority to 40_8 and creates a second task via call .QTSK at priority 30_8. The new task, called QUE, will be created and readied every three seconds. After creating task QUE, the main program momentarily retains CPU control and types a prompt (?) on the system console. The main program task recognizes two command: B) (return to the CLI) and C) (load overlay COMP and execute code in overlay COMP). Overlay COMP types the message:

I am a Data General Computer.

The code in overlay COMP then releases the overlay node and goes back to the prompt loop in the main program. (on characters other than B or C, the main program repeats the prompt loop.)

Very soon after the main program has typed its prompt, and while it is waiting for a B or C, the QUE task is readied. At the next device interrupt (from the Real Time Clock, console, etc.), rescheduling occurs and the Task Scheduler gives QUE CPU control because it has a higher priority than the main program. The system, under direction of the Task Scheduler, suspends the main program, loads the overlay containing QUE, and transfers control to code in QUE; QUE then types the message:

I'm the queued task...about to OVKIL myself.

At this point, QUE prints the prompt (?) and kills itself via call .OVKIL. This gives control back to the main program, which once again waits for a B) or C) . In three seconds, task QUE is created and readied again and the whole sequence repeats.

When QUE is ready to run, it gets control, types its message, and kills itself very quickly. In fact, because QUE issues system calls, it is suspended briefly before it can type the message and prompt -- this gives the main program a slice of CPU control. All this means that the person who runs the program can type B) or C) at any time and get a very fast response.

The two tasks (the main program and QUE) are totally unaware of one another. Furthermore, when an interrupt occurs and the scheduler decides to suspend one task and execute another, the original task simply continues from the point at which it was suspended -- which can be any location in its address space. When QUE kills itself, its whole state (TCB data) is wiped out; after the .QTSK interval, it is created as a brand-new task. Thus there is no simple way to have QUE return control to the prompt loop in the main program, which is why we have QUE type a prompt before it kills itself. (The .XMT and .REC calls could return control to the main prompt loop, but this would have produced a far more complex example.)

Here is a sample of dialog from EXAMPLE:

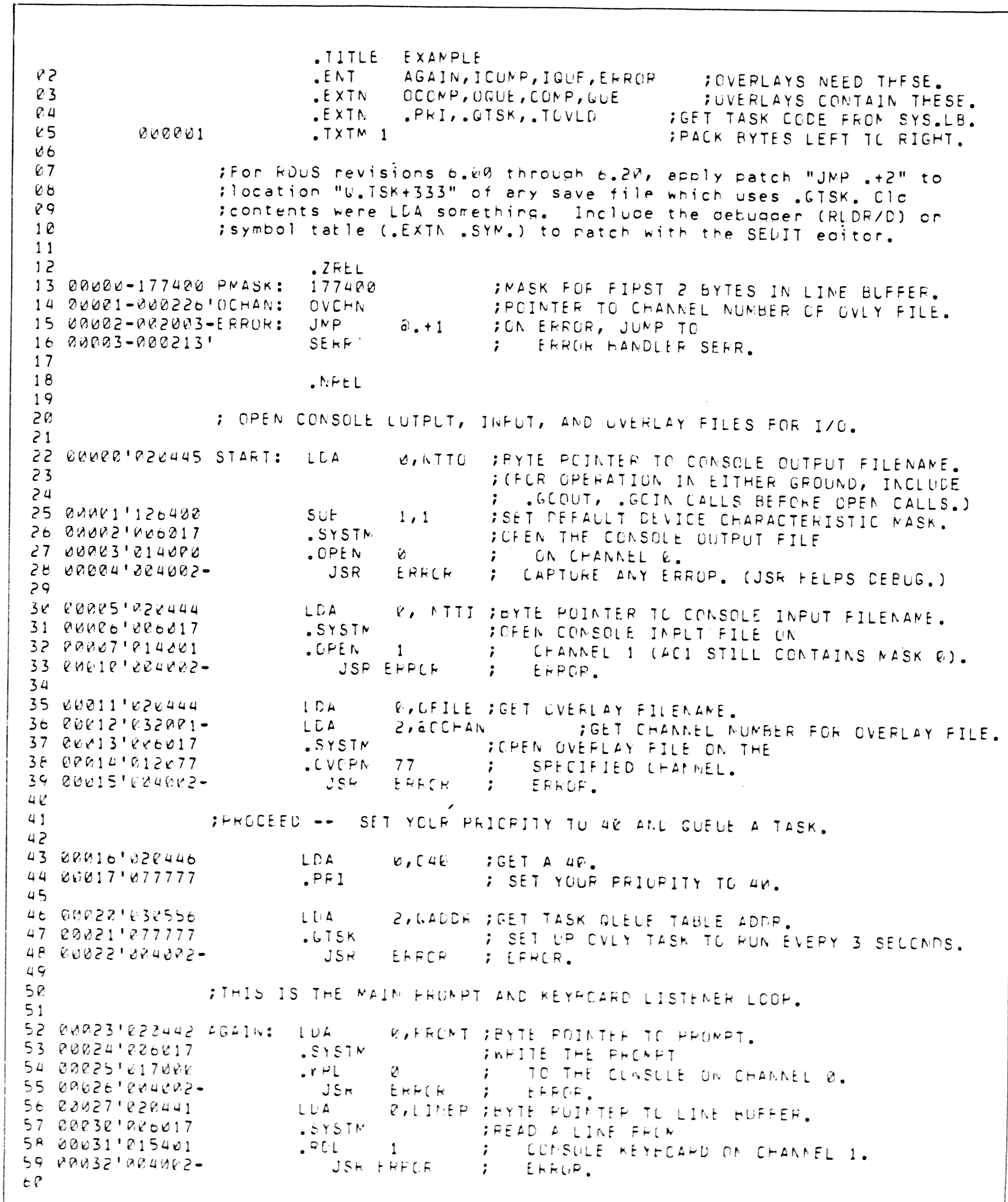
```
R
EXAMPLE)
?
I'm the queued task...about to OVKIL myself.
?
C)
I am a Data General computer.
?
I'm the queued task...about to OVKIL myself.
B)
R
```

The assembler command for the EXAMPLE program was:

```
MAC/L (EXAMPLE.QUE. COMP))
```

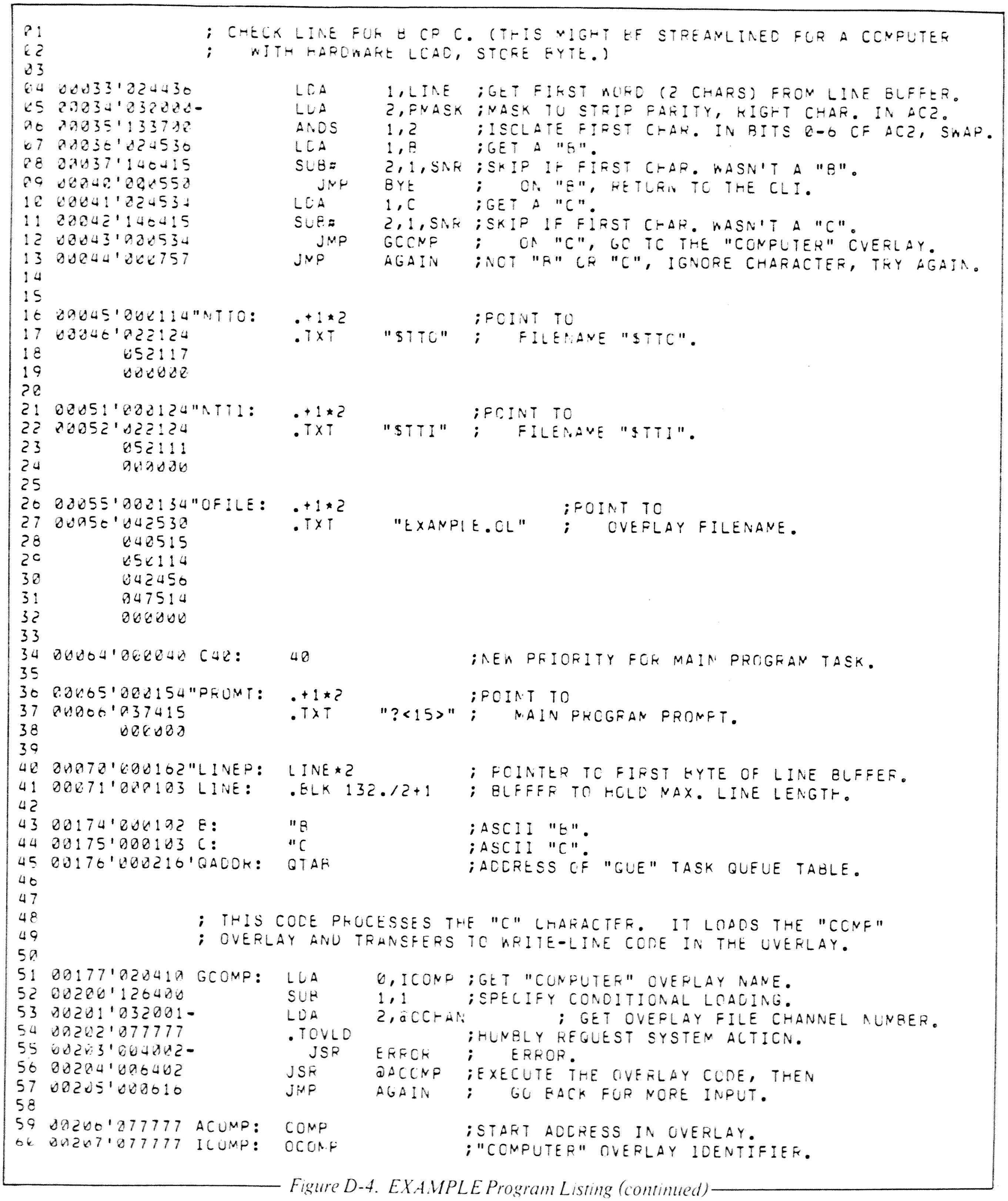
The load line was:

```
RLDR 2/K EXAMPLE [QUE.COMP])
```



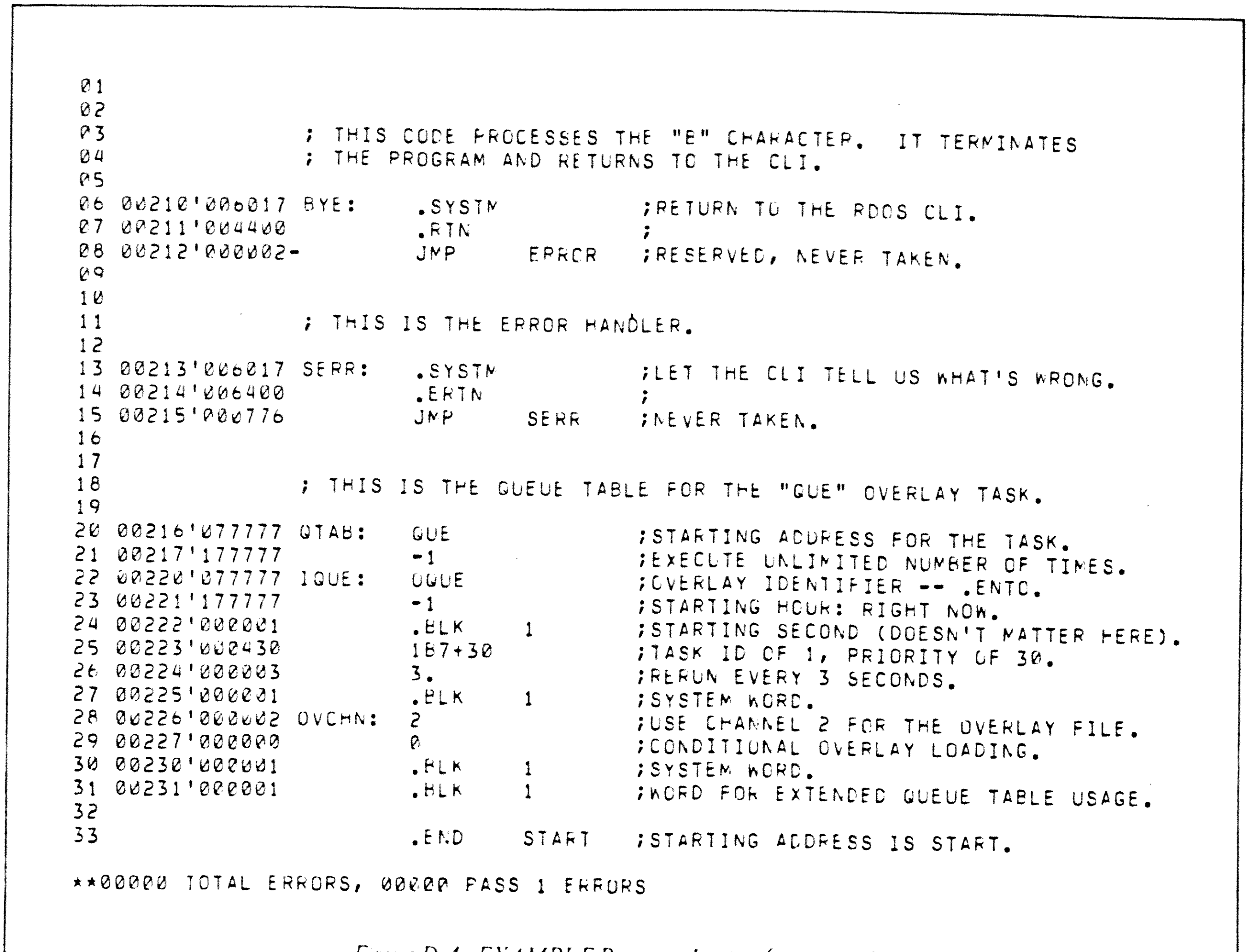
```
                         .TITLE  EXAMPLE
                         .ENT    AGAIN,ICOMP,IQUE,ERROR     ;OVERLAYS NEED THESE.
                         .EXTN   OCOMP,OQUE,COMP,QUE        ;OVERLAYS CONTAIN THESE.
                         .EXTN   .PRI,.GTSK,.TOVLD        ;GET TASK CODE FROM SYS.LB.
          000001         .TXTM 1                          ;PACK BYTES LEFT TO RIGHT.

                 ;For RDOS revisions 6.00 through 6.20, apply patch "JMP .+2" to
                 ;location "G.TSK+333" of any save file which uses .GTSK. Old
                 ;contents were LDA something.  Include the debugger (RLDR/D) or
                 ;symbol table (.EXTN .SYM.) to patch with the SEDIT editor.

                         .ZREL
 00000-177400 PMASK:     177400              ;MASK FOR FIRST 2 BYTES IN LINE BUFFER.
 00001-000226'OCHAN:     OVCHN               ;POINTER TO CHANNEL NUMBER OF OVLY FILE.
 00002-002003-ERROR:     JMP     @.+1        ;ON ERROR, JUMP TO
 00003-000213'           SERR                ;    ERROR HANDLER SERR.

                         .NREL

                 ; OPEN CONSOLE OUTPUT, INPUT, AND OVERLAY FILES FOR I/O.

 00000'020445 START:     LDA     0,NTTO  ;BYTE POINTER TO CONSOLE OUTPUT FILENAME.
                                         ;(FOR OPERATION IN EITHER GROUND, INCLUDE
                                         ;  .GCOUT, .GCIN CALLS BEFORE OPEN CALLS.)
 00001'126400            SUB     1,1     ;SET DEFAULT DEVICE CHARACTERISTIC MASK.
 00002'006017            .SYSTM          ;OPEN THE CONSOLE OUTPUT FILE
 00003'014000            .OPEN   0       ;    ON CHANNEL 0.
 00004'004002-             JSR   ERROR   ;  CAPTURE ANY ERROR. (JSR HELPS DEBUG.)

 00005'020444            LDA     0, NTTI ;BYTE POINTER TO CONSOLE INPUT FILENAME.
 00006'006017            .SYSTM          ;OPEN CONSOLE INPUT FILE ON
 00007'014001            .OPEN   1       ;    CHANNEL 1 (AC1 STILL CONTAINS MASK 0).
 00010'004002-              JSR ERROR    ;    ERROR.

 00011'020444            LDA     0,OFILE ;GET OVERLAY FILENAME.
 00012'032001-           LDA     2,@OCHAN        ;GET CHANNEL NUMBER FOR OVERLAY FILE.
 00013'006017            .SYSTM          ;OPEN OVERLAY FILE ON THE
 00014'012077            .OVOPN  77      ;    SPECIFIED CHANNEL.
 00015'004002-             JSR   ERROR   ;    ERROR.

                 ;PROCEED --  SET YOUR PRIORITY TO 40 AND QUEUE A TASK.

 00016'020446            LDA     0,C40   ;GET A 40.
 00017'077777            .PRI            ; SET YOUR PRIORITY TO 40.

 00020'030556            LDA     2,QADDR ;GET TASK QUEUE TABLE ADDR.
 00021'077777            .GTSK           ; SET UP OVLY TASK TO RUN EVERY 3 SECONDS.
 00022'004002-             JSR   ERROR   ; ERROR.

                 ;THIS IS THE MAIN PROMPT AND KEYBOARD LISTENER LOOP.

 00023'022442 AGAIN:     LDA     0,PRMPT ;BYTE POINTER TO PROMPT.
 00024'006017            .SYSTM          ;WRITE THE PROMPT
 00025'017000            .WRL    0       ;    TO THE CONSOLE ON CHANNEL 0.
 00026'004002-             JSR   ERROR   ;    ERROR.
 00027'020441            LDA     0,LINEP ;BYTE POINTER TO LINE BUFFER.
 00030'006017            .SYSTM          ;READ A LINE FROM
 00031'015401            .RDL    1       ;    CONSOLE KEYBOARD ON CHANNEL 1.
 00032'004002-              JSR ERROR    ;    ERROR.
```

Figure D-4. EXAMPLE Program Listing



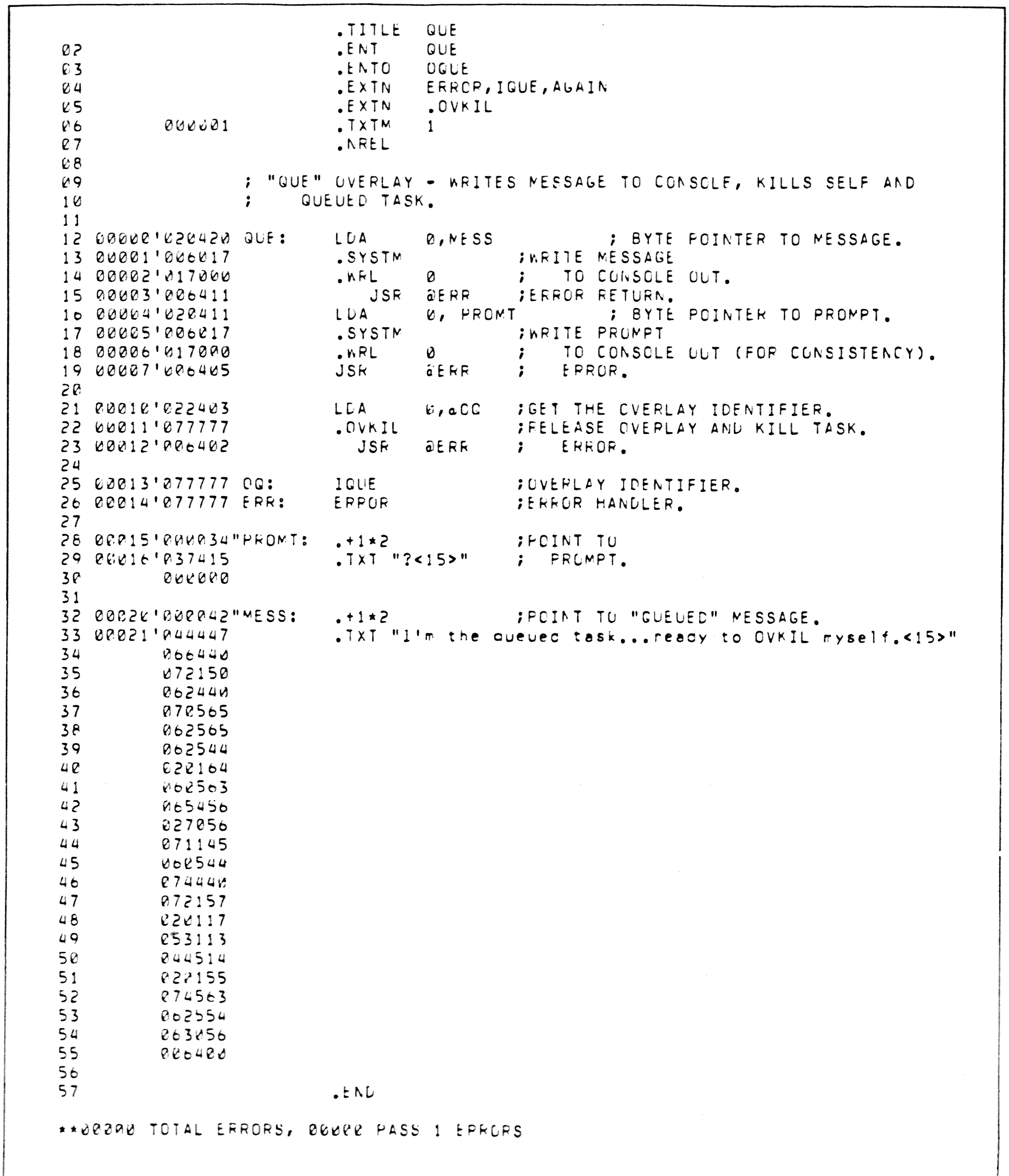
```
                        ; CHECK LINE FOR B OR C. (THIS MIGHT BE STREAMLINED FOR A COMPUTER
                        ;    WITH HARDWARE LOAD, STORE BYTE.)

00033'024436              LDA      1,LINE  ;GET FIRST WORD (2 CHARS) FROM LINE BUFFER.
00034'032000-             LDA      2,PMASK ;MASK TO STRIP PARITY, RIGHT CHAR. IN AC2.
00035'133700              ANDS     1,2     ;ISOLATE FIRST CHAR. IN BITS 0-6 OF AC2, SWAP.
00036'024536              LDA      1,B     ;GET A "B".
00037'146415              SUB#     2,1,SNR ;SKIP IF FIRST CHAR. WASN'T A "B".
00040'000550                JMP    BYE     ;   ON "B", RETURN TO THE CLI.
00041'024534              LDA      1,C     ;GET A "C".
00042'146415              SUB#     2,1,SNR ;SKIP IF FIRST CHAR. WASN'T A "C".
00043'000534                JMP    GCOMP   ;   ON "C", GO TO THE "COMPUTER" OVERLAY.
00044'000757              JMP      AGAIN   ;NOT "B" OR "C", IGNORE CHARACTER, TRY AGAIN.


00045'000114"NTTO:    .+1*2                ;POINT TO
00046'022124          .TXT     "$TTO"  ;   FILENAME "$TTO".
      052117
      000000

00051'000124"NTTI:    .+1*2                ;POINT TO
00052'022124          .TXT     "$TTI"  ;   FILENAME "$TTI".
      052111
      000000

00055'000134"OFILE:   .+1*2                        ;POINT TO
00056'042530          .TXT      "EXAMPLE.OL"   ;   OVERLAY FILENAME.
      040515
      050114
      042456
      047514
      000000

00064'000040 C40:     40                   ;NEW PRIORITY FOR MAIN PROGRAM TASK.

00065'000154"PROMT:   .+1*2                ;POINT TO
00066'037415          .TXT     "?<15>" ;   MAIN PROGRAM PROMPT.
      000000

00070'000162"LINEP:   LINE*2               ; POINTER TO FIRST BYTE OF LINE BUFFER.
00071'000103 LINE:    .BLK 132./2+1        ; BUFFER TO HOLD MAX. LINE LENGTH.

00174'000102 B:       "B                   ;ASCII "B".
00175'000103 C:       "C                   ;ASCII "C".
00176'000216'QADDR:   QTAB                 ;ADDRESS OF "QUE" TASK QUEUE TABLE.


                      ; THIS CODE PROCESSES THE "C" CHARACTER.  IT LOADS THE "COMP"
                      ; OVERLAY AND TRANSFERS TO WRITE-LINE CODE IN THE OVERLAY.

00177'020410 GCOMP:   LDA      0,ICOMP ;GET "COMPUTER" OVERLAY NAME.
00200'126400          SUB      1,1     ;SPECIFY CONDITIONAL LOADING.
00201'032001-         LDA      2,@CCHAN        ; GET OVERLAY FILE CHANNEL NUMBER.
00202'077777          .TOVLD               ;HUMBLY REQUEST SYSTEM ACTION.
00203'004002-           JSR    ERROR   ;   ERROR.
00204'006402          JSR      @ACOMP  ;EXECUTE THE OVERLAY CODE, THEN
00205'000616          JMP      AGAIN   ;   GO BACK FOR MORE INPUT.

00206'077777 ACOMP:   COMP                 ;START ADDRESS IN OVERLAY.
00207'077777 ICOMP:   OCOMP                ;"COMPUTER" OVERLAY IDENTIFIER.
```

Figure D-4. EXAMPLE Program Listing (continued)



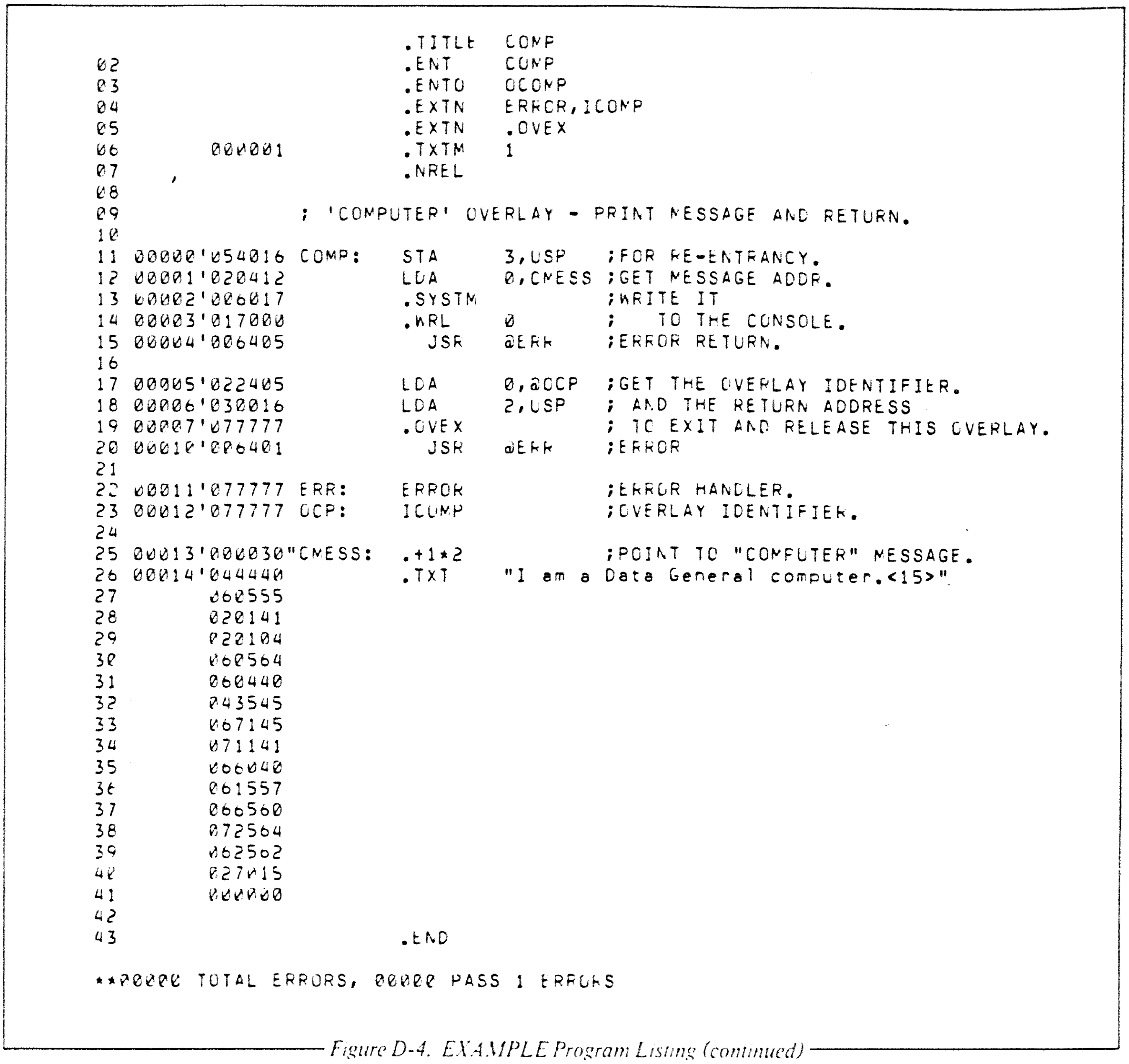
```
                         ; THIS CODE PROCESSES THE "E" CHARACTER.  IT TERMINATES
                         ; THE PROGRAM AND RETURNS TO THE CLI.

00210'006017 BYE:     .SYSTM              ;RETURN TO THE RDOS CLI.
00211'004400          .RTN                ;
00212'000002-         JMP      ERROR      ;RESERVED, NEVER TAKEN.


                         ; THIS IS THE ERROR HANDLER.

00213'006017 SERR:    .SYSTM              ;LET THE CLI TELL US WHAT'S WRONG.
00214'006400          .ERTN               ;
00215'000776          JMP      SERR       ;NEVER TAKEN.


                         ; THIS IS THE QUEUE TABLE FOR THE "QUE" OVERLAY TASK.

00216'077777 QTAB:    QUE                 ;STARTING ADDRESS FOR THE TASK.
00217'177777          -1                  ;EXECUTE UNLIMITED NUMBER OF TIMES.
00220'077777 IQUE:    QQUE                ;OVERLAY IDENTIFIER -- .ENTO.
00221'177777          -1                  ;STARTING HOUR: RIGHT NOW.
00222'000001          .BLK     1          ;STARTING SECOND (DOESN'T MATTER HERE).
00223'000430          1B7+30              ;TASK ID OF 1, PRIORITY OF 30.
00224'000003          3.                  ;RERUN EVERY 3 SECONDS.
00225'000001          .BLK     1          ;SYSTEM WORD.
00226'000002 OVCHN:   2                   ;USE CHANNEL 2 FOR THE OVERLAY FILE.
00227'000000          0                   ;CONDITIONAL OVERLAY LOADING.
00230'000001          .BLK     1          ;SYSTEM WORD.
00231'000001          .BLK     1          ;WORD FOR EXTENDED QUEUE TABLE USAGE.

                      .END     START      ;STARTING ADDRESS IS START.

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure D-4. EXAMPLE Program Listing (continued)



```
                           .TITLE  QUE
                           .ENT    QUE
                           .ENTO   OQUE
                           .EXTN   ERROR,IQUE,AGAIN
                           .EXTN   .OVKIL
         000001            .TXTM   1
                           .NREL

                      ; "QUE" OVERLAY - WRITES MESSAGE TO CONSOLE, KILLS SELF AND
                      ;     QUEUED TASK.

00000'020420 QUE:      LDA     0,MESS           ; BYTE POINTER TO MESSAGE.
00001'006017           .SYSTM           ;WRITE MESSAGE
00002'017000           .WRL    0        ;    TO CONSOLE OUT.
00003'006411              JSR  @ERR     ;ERROR RETURN.
00004'020411           LDA     0, PROMT         ; BYTE POINTER TO PROMPT.
00005'006017           .SYSTM           ;WRITE PROMPT
00006'017000           .WRL    0        ;    TO CONSOLE OUT (FOR CONSISTENCY).
00007'006405           JSR     @ERR     ;    ERROR.

00010'022403           LDA     0,@OQ    ;GET THE OVERLAY IDENTIFIER.
00011'077777           .OVKIL           ;RELEASE OVERLAY AND KILL TASK.
00012'006402             JSR   @ERR     ;    ERROR.

00013'077777 OQ:       IQUE             ;OVERLAY IDENTIFIER.
00014'077777 ERR:      ERROR            ;ERROR HANDLER.

00015'000034"PROMT:    .+1*2            ;POINT TO
00016'037415           .TXT "?<15>"     ;  PROMPT.
        000000

00020'000042"MESS:     .+1*2            ;POINT TO "QUEUED" MESSAGE.
00021'044447           .TXT "I'm the queued task...ready to OVKIL myself.<15>"
        066440
        072150
        062440
        070565
        062565
        062544
        020164
        060563
        065456
        027056
        071145
        060544
        074440
        072157
        020117
        053113
        044514
        020155
        074563
        062554
        063056
        006400

                           .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure D-4. EXAMPLE Program Listing (continued)



```
                             .TITLE  COMP
                             .ENT    COMP
                             .ENTO   OCOMP
                             .EXTN   ERROR,ICOMP
                             .EXTN   .OVEX
       000001                .TXTM   1
                             .NREL

                  ; 'COMPUTER' OVERLAY - PRINT MESSAGE AND RETURN.

00000'054016 COMP:    STA     3,USP    ;FOR RE-ENTRANCY.
00001'020412          LDA     0,CMESS  ;GET MESSAGE ADDR.
00002'006017          .SYSTM           ;WRITE IT
00003'017000          .WRL    0        ;    TO THE CONSOLE.
00004'006405            JSR   @ERR     ;ERROR RETURN.

00005'022405          LDA     0,@OCP   ;GET THE OVERLAY IDENTIFIER.
00006'030016          LDA     2,USP    ; AND THE RETURN ADDRESS
00007'077777          .OVEX            ; TO EXIT AND RELEASE THIS OVERLAY.
00010'006401            JSR   @ERR     ;ERROR

00011'077777 ERR:     ERROR            ;ERROR HANDLER.
00012'077777 OCP:     ICOMP            ;OVERLAY IDENTIFIER.

00013'000030"CMESS:   .+1*2            ;POINT TO "COMPUTER" MESSAGE.
00014'044440          .TXT    "I am a Data General computer.<15>"
       060555
       020141
       022104
       060564
       060440
       043545
       067145
       071141
       066040
       061557
       066560
       072564
       062562
       027015
       000000

                      .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure D-4. EXAMPLE Program Listing (continued)

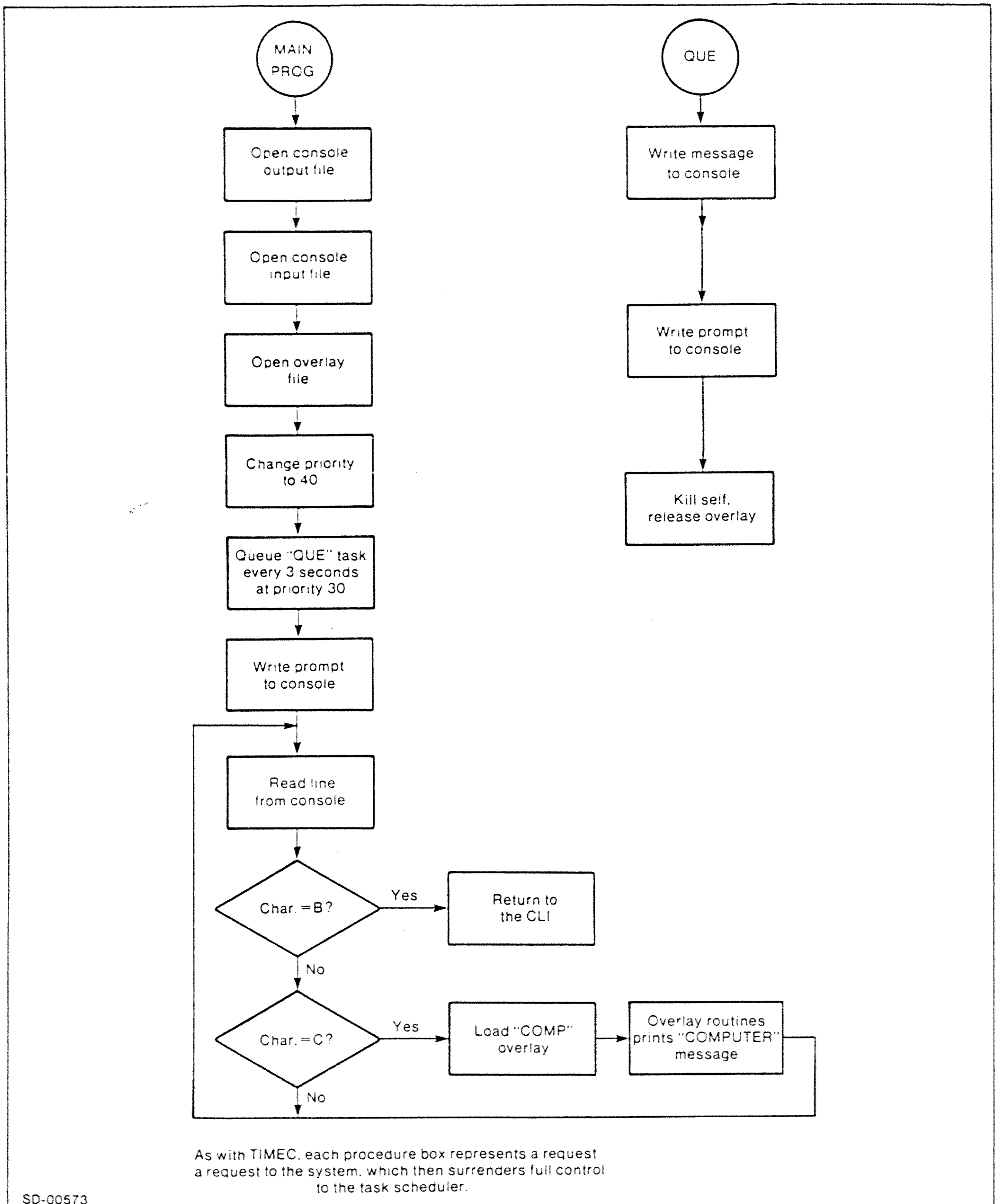


Figure D-5. EXAMPLE Flowchart.

End of Appendix

Appendix E
Overlay Directory Structure

When you load a program which has an associated overlay file, RLDR creates an overlay directory for it. During program execution, this directory occupies low NREL memory, right above the TCB pool, and it contains a four-word descriptor for each overlay. In a mapped system, the directory must fit into the lowest 1K block of memory.

You, or your program, can examine the directory through entry USTOD in the User Status Table. USTOD points to the directory base; it contains -1 if there are no overlays. The overlay directory built for each multitask program has the structure shown in Figure E-1.

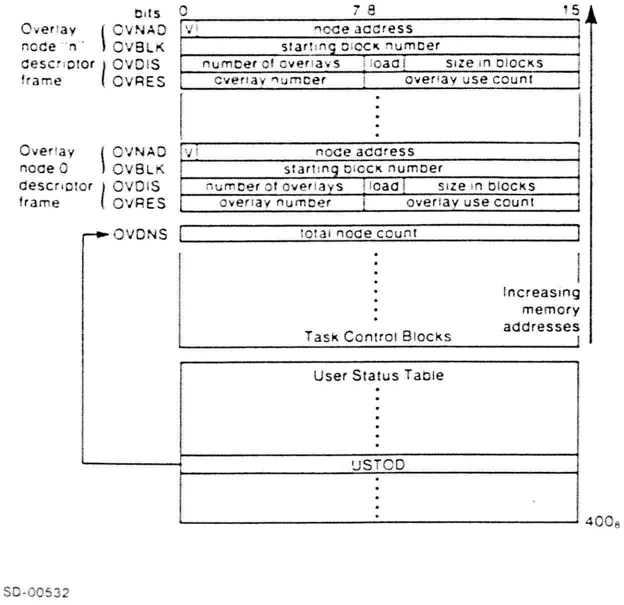


Figure E-1. Overlay Directory Structure (multitask)

Each overlay node in the save file has a corresponding four-word descriptor frame. Bits 0-7 of OVRES contain the number of the overlay which currently resides in the overlay node or which RDOS is loading into it. The overlay use count (OUC) (bits 8-15 of OVRES) describes the number of tasks using or requesting the resident overlay. RDOS uses OUC only in a multitask environment; see .TOVLD, Chapter 5.

Bits 0 to 7 of OVIDS describe the number of overlays associated with this overlay node (i.e., included in the same pair of square brackets in the RLDR command line). RDOS uses the load bit, bit 8, in multitask programs (.TOVLD). Bits 9 to 15 of this word describe the size (in integer multiples of 400_8 words, the size of each disk block) of this overlay node. OVBLK contains the starting logical disk block address of this node's segment in the overlay file, and OVNAD contains the memory address for the start of this overlay mode. For virtual overlay node, RDOS sets B0 of OVNAD to 1.

The overlay directory built for a single-task environment is identical to that described above except that the system ignores the load bit. A program can define a maximum of 256 overlay nodes in both multitask and single task environments. The maximum number of 256-word overlay nodes is 124 (which need about 60K bytes of memory). Page zero and task scheduler space requirements limit the maximum size of a single overlay to 126 disk blocks (64K bytes).

End of Appendix

Appendix F
Exceptional System Status

Certain serious error conditions can either halt the entire system in a crash, or cause the system to suspend processing and display an exceptional status or a trap message. The message returned from exceptional status or a trap will help identify the error; no information will return from a crash.

Both exceptional status and crash condition require full initialization of all disks that were initialized when the condition occurred.

Traps

A trap is less serious than an exceptional status or a crash; we have described traps here because they do stop program execution.

On a trap, the system displays the contents of the program counter and the accumulators on the console in this format:

```
TRAP (PC) (AC0) (AC1) (AC2) (AC3)
```

Bit 0 of the PC is carry.

In both mapped and unmapped systems, a trap usually results from a violation of map protection. The memory-image file (F)BREAK.SV is created and placed in the current directory.

For some user causes, see "Dual Programming - Mapped Systems," in Chapter 6, for details. Some common causes are: someone tried to access memory outside his logical space; or modify write-protected memory; or used more than 16 indirect references to an address; or tried to access a system device without having issued .DEBL (Chapter 3).

In some cases, the CLI will regain control after a trap, in others, not.

Exceptional Status

If you selected the core dump feature at SYSGEN, you can dump a core image of address space on the line printer, tape, or diskette after a crash or exceptional status. This dump is described below.

In exceptional status, the system will output the contents of the accumulators and an error code on the console, for example:

```
000015    177777    000011    037500    100010
AC0       AC1       AC2       AC3       error code
```

Note that if a SYSTEM error caused the exceptional status, bit 0 of the error code will be reset to 0, and the rest of the code word will contain a system error number (explained in Appendix A). The dump procedure described below applies to both kinds of error. If bit 0 is set to 1, the last two digits of the error code have the following meanings:

1 File system inconsistency detected; that is, RDOS tried to return a master device block which had no record in MAP.DR.

2 RDOS detected a SYS.DR error while accessing a directory on the master device. This means that either the entry count in a block of the directory exceeds 16_8, or a free entry in the block was indicated but RDOS could not find the free entry. If AC0 contains 16, AC2 contains the illegal count; if AC0 doesn't contain 16, RDOS expected a free entry but did not find it.

3 Interrupt stack overflow. The low-order bits of AC0 contain the address of the overflowed stack. If this is a system stack address (see load map), the cause can be a system device. If the address is not a system stack address, the cause is a software stack fault.

4 Inconsistent system data, such as an illegal device address. This will also happen if you INIT or DIR to a "new" disk before fully initializing it with INIT/F.

5 Master device data error; run a disk reliability test.

6 Master device timeout. If there are no obvious errors, run a disk reliability test.

7 Illegal device address on the moving-head master device. This can be caused by a misreading of the disk. Run a disk reliability test.

10 RDOS has detected an undefined interrupt and cannot clear it via an NIOC. The right byte of AC2 contains the code of the device.

12 There aren't enough contiguous disk blocks available to build push space indexes.

13 Attempted .RTN from level 0 in the background. Remove this instruction from the level 0 program, or execute it at a lower level.

14 Inconsistent IPB data. Perform an IPB reliability test. AC2 can give a clue to the problem, *if* the following conditions were true when the exceptional status occurred:

1. Both processors were up, and running the same revision of RDOS.
2. No user program issued I/O commands to the IPB, or overwrote the (unmapped) system.

- If AC2 contains -1 or a DCB address, the exceptional system status indicates an internal system (software) bug.
- If AC2 has a cell address, then RDOS has received an invalid message type; AC1 has the type byte. This problem indicates an IPB hardware failure.
- If AC2 has an address in the IPB interrupt handler (between IPBDC and IVTINT), then this is the address at which the exceptional system status actually occurred. If AC0 and AC1 do not contain 64400_8, then the interrupt handler detected an invalid condition, such as incorrect message length. This indicates an IPB hardware failure. If AC0 and AC1 equal 64400_8, then that processor timed out to the other processor, but resumed communication without booting. This would happen if the operator pressed the STOP switch and more than one and a half seconds later pressed CONTINUE; or if a user program turned interrupts off for more than one and one-half seconds (e.g., via the interrupt-disable debugger).

15 A hardware map violation (trap) occurred while a user interrupt rountine or user clock had control. The AC0 data field output on the console will contain the PC, not the contents of AC0, in this exceptional status.

16 ECLIPSEs with ERCC option only. Multibit ERCC memory error. See appropriate CPU technical manual.

17 NOVA 3s with hardware parity option only: Hardware parity error.

20 (INFOS system only) insufficient memory available at initialization time.

21 The spooler detected a MAP.DR error.

Controlling Exceptional Status

If you have an unmapped system, you can write your own routine to handle exceptional status situations. Your programs must store the address of your routine in location 11_8, at run time, and restore the original value before the program ends. Your routine will then gain control at an exceptional status; the console will not display the accumulator/error code message, but AC0, AC1, and AC2 will retain the contents they had at the error, and AC3 will contain the address of the error code.

If you have a mapped system, you must modify the operating system at source level if you want to insert your own exceptional status routine.

Producing a Core Dump

When this RDOS system was generated, the person who generated it determined whether or not you can produce a dump and which device will receive the dump. A SYSGEN question asks about the CORE DUMP FACILITY, and the answer given was 0 (no dump), 1 (line printer dump), 2 (magnetic tape dump), or 3 (diskette dump). If you are dumping after an exceptional status, proceed to the appropriate section below, and execute the steps there. If you are dumping after a crash, take the following steps.

After a system crash, the console will display nothing. Press the CPU switch STOP, then record the contents of the ACs, PC, carry, and the machine state. Now, lift RESET, and enter 11_8 in the data switches, lift EXAMINE, and note the number returned in the data lights. Enter this number in the data switches, lift RESET, then START. The console will then display the contents of the accumulators and an error code:

```
nnnnnn nnnnnn nnnnnn nnnnnn eeeeee
```

Disregard the error code, and proceed with the sequence described above for exceptional status dumps.

Line Printer Dump

In this dump, you can select portions of memory, or dump all of memory.

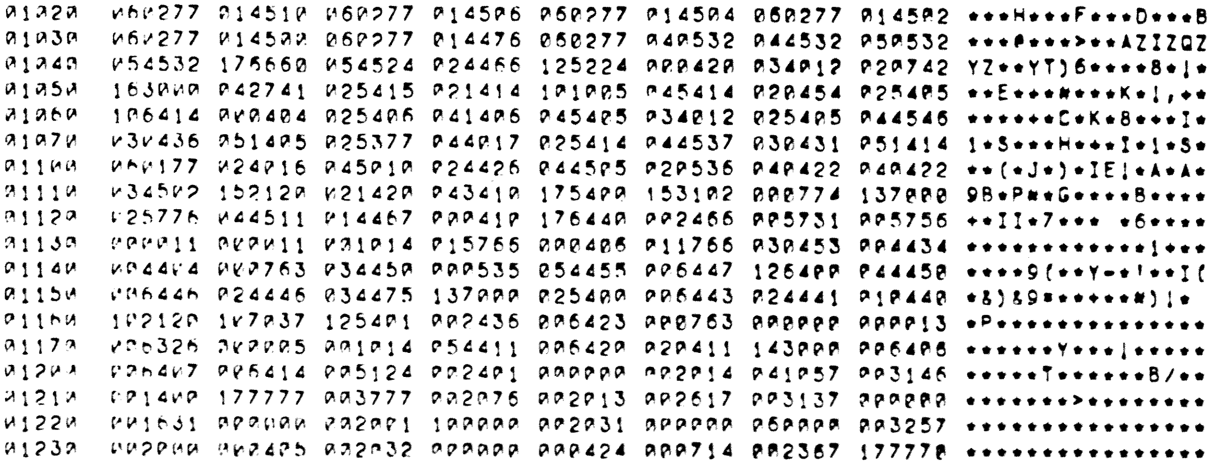
The line printer dump has three parts: the left column shows a memory address, the middle 8 columns show the contents of each word in the address, and the right column shows the ASCII value (if any) of each byte in the address. Figure F-1 contains a sample line printer dump.

To dump the entire address space of either a mapped or unmapped machine, press CONTINUE twice. To dump selected portions of address space, follow one of these procedures:

Unmapped Machines: Load the desired starting dump address into the data switches. Press CONTINUE; the CPU will halt. Load the desired ending address of the dump into the data switches, and again, press CONTINUE. You can enter as many starting/ending address pairs as you wish.

Mapped Machines: RDOS will shift each address that you input via the data switches left three bits so that you can dump the full range of possible mapped addresses. That is, if data switch 15 is up, and the rest down, this will be interpreted as address 10_8; this adds an implicit zero to any address you enter. To dump a range of mapped addresses, load the desired starting dump address into the data switches, and press CONTINUE. The CPU will halt. Load the desired ending address of the dump into the data switches, and again, press CONTINUE. RDOS will dump all locations from the low order address (times 10_8) to (but not including) the high order address (times 10_8). That is, if you select low address *l* and high address *h* on the data switches, RDOS will dump locations 10_8 *l through (10_8 *h) -1. Repeat the dumping process as often as you wish.

You can abort the core dump anytime by striking any key on the console and proceed with another dump sequence as you desire.



```
01020   060277 014510 060277 014506 060277 014504 060277 014502 ***H***F***D***B
01030   060277 014500 060277 014476 060277 040532 044532 050532 ***@***>**AZIZQZ
01040   054532 176660 054524 024466 125224 000420 034012 020742 YZ**YT)6****8*|*
01050   163000 042741 025415 021414 101005 045414 020454 025405 **E***#***K*|,**
01060   106414 000404 025406 041406 045405 034012 025405 044546 ******C*K*8***I*
01070   030436 051405 025377 044017 025414 044537 030431 051414 1*S***H***I*|*S*
01100   000177 024016 045010 024426 044505 020536 040422 040422 **(*J*)*IE|*A*A*
01110   034502 152120 021420 043410 175400 153102 000774 137000 9B*P#*G****B****
01120   025776 044511 014467 000410 176440 002466 005731 005756 **II*7*** *6****
01130   000011 000011 031014 015766 000406 011766 030453 004434 ************|***
01140   004404 000763 034450 000535 054455 006447 126400 044450 ****9(**Y-*!**I(
01150   006446 024446 034475 137000 025400 006443 024441 010440 *&)&9******#)|*
01160   102120 107037 125401 002436 006423 000763 000000 000013 *P**************
01170   006326 300005 001014 054411 006420 020411 143000 006406 ******Y***|*****
01200   026407 006414 005124 002401 000000 002014 041057 003146 *****T******B/**
01210   001400 177777 003777 002076 002013 002617 003137 000000 ********>*******
01220   001631 000000 002001 100000 002031 000000 060000 003257 ****************
01230   002000 002405 032032 000000 000424 000714 002367 177770 ****************
```

Figure F-1. Sample Line Printer Dump

Magnetic Tape Dump

To dump to magnetic tape, follow these steps:

1. Select unit number 3 on a magnetic tape drive, and make sure no other drive has this number. Mount a blank tape (300' or more), with ring in, on this drive. Then press drive switches LOAD and ON LINE.

2. Press the CPU switch CONTINUE. The dump program then displays the message READY?.

3. Press the CPU switch CONTINUE again. The dump program then copies all memory address to the tape, and displays the message DONE, then READY on the console. To stop the program, press CPU switch STOP; to produce another dump, RESET and UNLOAD the tape with drive switches, mount another tape and execute steps 2 and 3 again.

4. If you have forgotten a step, the program displays the message ERROR, then READY?. Execute the step and press the CPU switch CONTINUE.

The magnetic tape cannot be copied under RDOS.

Diskette Dump

To dump to diskette, follow these steps:

1. Select unit number 3 on a diskette drive, and make sure no other drive connected to this controller has the same number.

2. Tape the write-protect hole of a Data General diskette (or other diskette which has been hardware formatted); insert this diskette in the drive. Shut the door and turn the diskette drive ON.

3. Press CPU switch CONTINUE. The dump program then displays the message READY?.

4. Press the CPU switch CONTINUE again. The dump routine copies memory to the diskette; if it displays the messages DONE and READY, go to step 9.

5. If all addresses won't fit on one diskette, the program displays the message REPLACE, then READY?. Open the diskette door, remove the diskette, insert another hardware-formatted diskette in the drive, and close the door. Press the CPU switch CONTINUE. The program then copies the rest of memory to the second diskette, and displays the message DONE and READY?.

6. The diskette dump is complete. To stop the program, press the CPU switch STOP; to produce another dump, remove the diskette, then execute steps 3, 4, and 5 again.

7. If you have forgotten a step, the program displays the message ERROR, then READY?. Execute the step and press CONTINUE.

The diskette dump cannot be copied under RDOS.

End of Appendix

Appendix G
Bootstrapping RDOS from Disk

This appendix describes the steps you must follow to bootstrap (start up) RDOS from disk.

The disk from which you bootstrap must be in the first drive on its controller: DP0, DP4, DZ0, DZ4, DS0, DS4, and so on. It must have a bootstrap root and a copy of BOOT.SV on it, but it need not have an RDOS system or CLI if another disk on your system has these on it.

The preliminary steps are:

1. Turn on the system console and any other CRT or printing consoles which the system will service. If the system console is an upper- and lowercase console, put it in ALPHA LOCK, because BOOT.SV does not accept lowercase letters.

2. Power up your computer by turning the POWER switch to ON.

3. Get your disk(s) ready. If you removed a removable disk cartridge or pack after your last RDOS session, insert it in its drive. If you have a nonremovable disk, or if you left a removable disk in its drive for convenience, proceed.

4. Press the rocker switch on the disk drive to READY, START, or RUN, depending on your type of disk. Wait for the READY light.

5. Now, with the preliminaries done, you can bootstrap the system. If your computer has a programmed console (i.e., a microcoded virtual console, as in the NOVA 4 family), go to step 6. If it has hardware data switches and automatic program load, go to step 7. If it lacks automatic program load, go to step 8.

6. The system console (CRT and printer) should show an exclamation point (!) prompt. Check Table G-1 for your disk device code, then type:

   1000nnL

   on the console and go to step 9.

7. Make sure the data switches are set as shown in Table G-1. Lift the RESET switch, then the PROGRAM LOAD switch, and go to step 9.

Table G-1. Disk Controller Device Codes

Disk Type	nn = (octal)	With hardware data switches, set these switches up (others down)
Fixed-head		
Model 6063-6064		
Controller # 1	26	0, 11, 13, 14
Controller # 2	66	0, 10, 11, 13, 14
Model 6001-6008		
Controller # 1	20	0, 11
Controller # 2	60	0, 10, 11
Moving-Head		
Model 6060-6061, 6067		
Controller # 1	27	0, 11, 13, 14, 15
Controller # 2	67	0, 10, 11, 13, 14, 15
All other disks		
Controller # 1	33	0, 11, 12, 14, 15
Controller # 2	73	0, 10, 11, 12, 14, 15

8. Key in a loader program via the switches:

   a. Set the data switches to 000376_8 (switches 8 through 14 up, the others down); then lift EXAMINE.

   b. Set the data switches to 0601nn (get nn from Table G-1). Lift DEPOSIT.

   c. Set the switches to 000377_8 (switches 8 through 15 up, the others down). Depress DEPOSIT NEXT.

   d. Set the data switches to 000376_8 (put down switch 15.) Lift RESET, then START.

9. Your program load steps will read the bootstrap root in from the beginning of the disk. The root then invokes BOOT.SV, and BOOT.SV asks:

   FILENAME?

   Respond with the name of your RDOS system, or of any other stand-alone program you want to execute. DKINIT.SV, BOOT.SV, or an RTOS or RDOS system is a stand-alone program. For example, type:

   ```
   MYSYS)
   ```

   to bootstrap a system named MYSYS.SV on the bootstrap disk. If the system or program you want is on a different disk (not the one you are bootstrapping from), precede its name with a *directory specifier*. A directory specifier is simply the name of the disk (directory) that holds the system, followed by a colon; e.g., DP0F: . Thus your response to the FILENAME? query might be:

   ```
   DP0F:MYSYS)
   ```

If your system name is SYS, and it is on the disk from which you are bootstrapping, you can simply type) in response to FILENAME?, because SYS is the default name.

10. RDOS will start up, asking you for the data and time:

    type RDOS REV x.xx
    DATE (M/D/Y) 1 10 79) (current date)

11. *TIME (H:M:S)?* 13 10) (current time)

    R

    When you see the CLI's R prompt, your RDOS system is ready to execute your commands. At this point, you may want to turn or press the computer power switch to the LOCK position; this disables all other front panel switches and prevents anyone from inadvertantly stopping RDOS by pressing them.

During the bootstrap, the disk directory that holds the RDOS system files becomes the master directory.

Before you turn off power to your computer or disks, be sure to RELEASE the RDOS system. You can do this either by typing the master directory name; e.g.,

```
RELEASE DP0)
```

or with the CLI variable %MDIR%, which contains the master directory name; e.g.,

```
RELEASE %MDIR%)
```

After either command, RDOS should display a sign-off message and shut down:

MASTER DEVICE RELEASED

If the foreground program is still running, you can terminate it with CTRL-F from the background console (STTI); if any system spool files are active, you can kill spooling to the appropriate device(s) with the SPKILL command. Then, type the RELEASE command again.

End of Appendix

Appendix J
Advanced Multitask Programming

For most multitask application programs, the features described in Chapter 5 will suffice. You need read this appendix only if:

- You want to write your own multitasking primitives (task calls)

- Your tasks require one or more special resources (for example, floating-point hardware), that the system does not provide for in a TCB.

The features described in this appendix can:

- provide more programming flexibility than the standard features alone, without requiring you to modify the task monitor sources; and

- provide this flexibility in a system-independent way. You can use the calls in this appendix to develop application programs for any system configuration (RDOS or RTOS, mapped or unmapped). All you need do to reconfigure for a different system is load a program (via RLDR) with the appropriate system libraries.

Before you proceed, you should be familiar with the material in Chapter 5.

Definitions

The following definitions relate to tasks and task states; they apply throughout RDOS and RTOS.

General Terms

Task Resources are those storage elements of the computer, such as accumulators and special memory locations, two or more tasks must share. The task scheduler allows such sharing by ensuring that the proper values for each task's resources appear in the actual storage elements of the computer while the task is executing. When a task is not executing, the current values of its resources are held in its TCB.

Rescheduling is the process of selecting and executing the highest priority ready task. The task scheduler performs rescheduling after each task call, after receiving control from the system following an interrupt, and when a system call completes. You can suppress rescheduling via the .DRSCH or .SINGL task calls, or by entering scheduler state, as described below. If you have not disabled rescheduling, you must assume that it can happen at any time.

A *task swap* occurs during rescheduling when the task scheduler determines that it should execute a different task from the one which was last executing. If the task which was executing was not terminated (by .KILL, etc.), the scheduler saves the current state of the task's resources in the task's TCB. The scheduler then restores the former state of the new task's resources from *its* TCB. Then, the scheduler places the new task's TCB in the active TCB chain at the end of its priority class, so that the next time rescheduling occurs the task will be considered for execution only after all other tasks in its class. Finally, the new task receives CPU control and becomes the current task.

CTCB is a location maintained by the scheduler which contains the address of the current task's TCB. If no task is currently active (for example, all tasks are suspended or rescheduling is occurring), CTCB contains the address of the most recent task's TCB, if that task was not terminated. If it was terminated, then CTCB contains 0. Thus CTCB identifies the task to which the current values of task resource storage elements belong; 0 means that these values are no longer relevant.

CTCB is a page zero location. You can access it as follows to obtain the TCB address for the current task:

```
.EXTD CTCB
LDA ac, CTCB
```

Location USTCT in the User Status Table (UST) also contains the current TCB address. However, you should use CTCB instead of USTCT.

The *hardware stack* is the storage element of the computer which has built-in stack functions. On an ECLIPSE computer, the hardware stack occupies locations 40_8 through 43_8. On a NOVA 3 or microNOVA computer, the stack occupies the stack and frame pointers and location 42_8, which the system interprets as the stack limit. RDOS treats the hardware stack as a task resource, thus it is available for use by all tasks.

A *reentrant* section of code (sequence of instructions) allows another task to enter this code before the original task exits. Code which several tasks can access is reentrant only if each task has its own local storage, which no other task executing the code can access. Giving each task its own stack area and using the stack for local storage is a common way to achieve reentrancy.

State Definitions

User state is the normal state for an application program. This is the state from which system and task calls are made, as described in Chapter 5. Code must be reentrant in user state if more than one task will use it. In this state, task execution is suspended on an interrupt if a higher priority task is ready for execution. A task in user state can use the User Stack Pointer (USP) and the hardware stack; it can also examine (but not modify) CTCB and the current TCB. In dual-ground operation, it can determine the current ground by examining USTPC in the UST; USTPC contains 0 for the background, or 1 for the foreground. If there are no indicators of other states, the program is in user state.

Singletask State is used occasionally for a critical section of an application program. You enter this state via the .SINGL task call; it prevents other tasks from gaining control. However, interrupts and the other ground (if any) continue to execute. A task can issue system calls from singletask state as well as from user state; it can also issue any task call except .MULTI or one which would kill or suspend itself. If it issues .MULTI, or kills or suspends itself, the program enters user state. Code executed from singletask state need not be reentrant. It can use USP, the hardware stack, CTCB, and the current TCB as it can in user state. If location SM.SW contains a nonzero value, the program is in singletask state.

Scheduler state is the normal state for task call code. An interrupt can cause temporary loss of control, but, unlike user and singletask states, control returns to the point of interruption without rescheduling. Thus, scheduler state ensures that no other task in the same ground will get control, although interrupts and the other ground continue. Code executed in scheduler state need not be reentrant. It should not use USP or the hardware stack; but it can both read and modify CTCB and the current TCB, subject to restrictions described later. In unmapped RDOS systems, code cannot use USTPC to distinguish foreground from background; instead, it should compare the UST base (USTAD) to the value 400_8. A value of 400_8 for USTAD indicates the background; a value other than 400_8 indicates the foreground. A task is in scheduler state for unmapped RDOS if location USTPC contains a value other than 0 or 1, or, for mapped RDOS, if location 1 is nonzero. For RTOS, location .SYS. is nonzero in scheduler state.

Use *interrupt-disabled state* to perform critical manipulation of TCB data or the active TCB chain. There is no way for a task in this state to lose control of the CPU, even temporarily.

Coding Your Own Task Calls

TCB and Status Bits

Two status bits of word TPRST in a TCB are allocated for your use; you can use them to extend the standard features. Bit TSUPN, the user suspend bit, will prevent a task from running when set. Bit TSUSR, the user status bit, will not affect task readiness but is available for storing an additional piece of task-related information.

Also, word TELN is available for your own use. A typical use for TELN is to store the address of a "TCB extension" in it. This allows you to store as much additional task-related information as you need.

Scheduler Calls

The scheduler calls defined below are, like task calls, external symbols which you must identify as .EXTN in your source program. The relocatable loader (RLDR) resolves them at load time, according to system type. Each version of SYS.LB defines the calls for its version of the system (RDOS or RTOS, mapped or unmapped, NOVA computer or ECLIPSE computer).

Enter Scheduler State (EN.SCHED)

To enter scheduler state from user or singletask state:

```
;AC3 not equal to 0 (also not equal to 1
;       for unmapped RDOS systems).
EN.SCHED
;Returns here with all ACs and carry preserved.
```

A task already in scheduler state can safely reissue EN.SCHED, but no change in state will occur.

Task State Save (.TSAVE)

For a task in scheduler state, this call saves the ACs, carry, and program counter in its TCB. The PC saved is the value in bits 1-15 of AC3 at the time of the last EN.SCHED.

```
;ACs, carry, PC to be saved.
.TSAVE
;Returns here with ACO, AC1, and carry unchanged,
;        AC2 = value that was in AC3 at
;        time of last EN.SCHED;
;        AC3 = TCB address.
```

EN.SCHED and .TSAVE are meant to be used together at the start of code which implements a user-designed task call. For a task call with error return, you might use them this way:

```
           .ENT .TASK, T.ASK
           .EXTN EN.SCHED, .TSAVE
           .ZREL
.TASK=     JSR @
           T.ASK
           .NREL
T.ASK:     INC 3,3       ;Assume normal return.
           EN.SCHED      ;Enter scheduler state.
            TSAVE        ;Save task state.
```

For a task call without an error return, you would omit the INCrement instruction.

Leave Scheduler State Normally (RE.SCHED)

When you successfully complete the processing for a task call, issue RE.SCHED to exit to the scheduler for rescheduling. Use RE.SCHED in scheduler state.

```
;No input.
RE.SCHED
;No return.
```

Leave Scheduler State Abnormally (ER.SCHED)

When you detect an error during task call processing, place an error code in AC2 and exit to the scheduler via ER.SCHED. This returns control to the location *preceding* the one specified by TPC, and passes back the error code in AC2. Use ER.SCHED in scheduler state.

```
;AC2 = error code.
ER.SCHED
;No return.
```

Enter Interrupt-Disabled State (INT.DS)

Use INT.DS to enter interrupt-disabled state from scheduler state.

```
;No input.
INT.DS
;Returns here with ACO, AC1, AC2,
; and carry unchanged.
```

Leave Interrupt-Disabled State (INT.EN)

To leave interrupt-disabled state and return to scheduler state, use INT.EN.

```
;No input.
INT.EN
;Returns here with ACO, AC1, AC2,
; and carry unchanged.
```

Task ID Search (ID.SRCH)

Use ID.SRCH to search for a task with a given ID. You can issue ID.SRCH in either scheduler or interrupt-disabled state.

```
;AC1, right byte=ID of sought task.
ID.SRCH
;Error return here, AC2 = error code.
;Normal return here, with AC2 = TCB address
; of sought task.
```

For both returns, AC0 and carry are preserved-- the left byte of AC1 is zeroed and the right is preserved.

Handling Additional Task Resources

This section tells you how to manage task resources that are not automatically managed by the system. At certain points in its scheduling process, the scheduler calls out to routines which you may supply to handle your additional task resources. These call-outs are described in the first section, below.

If floating-point hardware and/or a block of contiguous memory locations are among the resources you need, you can simply use a handler supplied in SYS.LB. This is explained in the second section, below.

If you want to handle additional task resources while using operator communications, see the final section.

Task Scheduler Call-outs

To use any call-out described below, write, assemble, and load a routine of the appropriate name and function. You must insert the name of the routine in the RDLR command line before RLDR searches SYS.LB (by default, this occurs at the end of the command line). If you do not supply a routine, RDLR will load a dummy routine, which does nothing, from SYS.LB.

Task Initiation Call-out (TSK.X)

This call-out allows you to endow a new task with additional task resources. When the scheduler initiates a task, it first removes a TCB from the free TCB chain. Then it initializes certain parts of the TCB, as described later under *Task Control Block Values*. The scheduler then calls out to your TSK.X routine in scheduler state.

Your TSK.X routine can initialize certain parts of the TCB and change the parts the scheduler initialized (subject to the restrictions mentioned in the TCB Values section). On a normal return, the scheduler links the TCB for the new task, as modified by your TSK.X code, into the active TCB chain.

The scheduler transfers control to address TSK.X with the accumulators set up as follows:

AC0 contains the value passed to .TASK in AC2. AC0 is irrelevant if .QTSK initiates the task.

AC1 contains -1 if .TASK initiates the task, or the address of the task queue table if .QTSK initiates the task.

AC2 contains the address of the TCB for new task.

AC3 contains the (error) return address.

The routine you supply with entry address TSK.X need not preserve accumulators or carry. If you detect an error, place an error code in AC2 and return control to the location whose address you received in AC3. On a normal return, return control to the location whose address is one greater than the one you received in AC3. For example:

```
              .ENT TSK.X
              .NREL
TSK.X:        STA 3, RTNAD      ;SAVE RETURN.
                  .
                  .
              COM# 1,1,SZR      ;.TASK OR .QTSK?
              JMP QUE
TSK:              .             ;HANDLE .TASK CASE.
                  .
                  .
QUE:              .             ;HANDLE .QTSK CASE.
                  .
                  .
BAD:          LDA 2, CODE       ;ERROR RETURN.
              JMP @RTNAD

GOOD:         ISZ RTNAD         ;NORMAL RETURN.
              JMP @RTNAD

RTNAD:        .BLK 1
```

When you return an error indication, and .TASK is initiating the task, the task will not be initiated, and its TCB will return to the free TCB chain; the error code you place in AC2 will be passed to the task which issued .TASK. When you return an error indication and .QTSK is initiating the task, the system will try again one second later.

Task Termination Call-out (TRL.X)

The TRL.X call out frees a task's additional resources when a task is terminated-- typically those resources you assigned in a TSK.X routine. The scheduler calls this routine in scheduler state whenever a task is being killed, with the task's TCB already unlinked from the active chain but not yet restored to the free chain. The scheduler transfers control to address TRL.X with ACs set up as follows:

AC2	contains the TCB address of task being killed.
AC3	contains the return address.

The routine you supply with entry address TRL.X need not preserve accumulators or carry. When you have finished your processing, return control to the location whose address you received in AC3. There is no way to signal an error from TRL.X.

Task Swap Call-out (ESV.X)

This call out allows you to save and restore additional task resources as needed when a task swap occurs. The scheduler calls the routine in scheduler state and transfers control to address ESV.X with the accumulators set up as follows:

AC2	TCB address for task losing control, or 0 if no task is losing control (as described under CTCB, earlier).
CTCB	TCB address of task gaining control.
AC3	return address.

The routine you supply with entry address ESV.X need not preserve accumulators or carry. When you have finished processing, return control to the location whose address you received in AC3. There is no way to signal an error from ESV.X.

A 0 passed to you in AC2 indicates that there is no valid most recently active task whose resources you would need to save. This situation occurs as the default task is initially selected for execution. ESV.X will be called with 0 in AC2 and the TCB address for the default task in CTCB. It also occurs after a task is terminated, because the terminated task's resources were freed by TRL.X, and are no longer meaningful to ESV.X.

Additional Resource Handler

The system library (SYS.LB) contains an ESV.X routine, which partly provides for the additional task resources of floating-point hardware and a block of contiguous storage words. To load this module, insert a .EXTN ESV.X in any source module whose name will occur in the RLDR command line before SYS.LB is searched.

For each task that needs access to the floating-point hardware, you must provide a block of words to store the task's values for its floating point state. The size and contents of this block depend on the kind of computer you use. For an ECLIPSE computer, the block has this format:

Status	2 words
FPAC0	4 words
FPAC1	4 words
FPAC2	4 words
FPAC3	4 words

This format matches that used by the FPSH and FPOP instructions.

For a NOVA computer, the format is:

FPAC	4 words
TEMP	4 words
Status	1 word

To provide for a block of contiguous memory locations as an additional task resource, you must define two symbols with .ENT and give them the following values:

ESV.S	equals the starting address of the block.
ESV.Z	equals the number of words in the block.

Also, you will need to provide a block of memory that is ESV.Z words long for each task that is to use this additional resource.

Finally, for each task that will use either the floating-point hardware or contiguous memory locations, you must initialize offset TELN in the TCB, within the TSK.X routine that you must supply. The value you place in TELN depends on the task's needs.

- If TELN contains either 0 or 100000_8, neither the floating-point resource nor the contiguous memory resource will be handled. Thus, since RDOS initializes TELN to 0, you need not change it for a task which needs neither resource.
- If the task requires floating-point hardware but not the contiguous memory, set TELN to the indirect address of the appropriate floating-point block described earlier.
- If the task needs the contiguous memory but not the floating-point hardware, set TELN to the (direct) address of a block of words ESV.Z+1 words long, and set the first of these words to either 0 or 100000_8. The contiguous memory locations will be saved in the remaining words of this block.

- If the task requires both resources, set TELN as immediately above, but set the first word of the block to the (direct) address of a floating-point save area as described above.

When you initialize TELN, you can also initialize the contents of these save areas as well. The values that your TSK.X routine places in these areas will be the initial values when the task being initiated starts executing.

Restrictions and Warnings

The system-supplied additional-resource handler assumes that TELN is set up properly for either or both of the resources; it does not prevent an unprepared task from using one of these resources inadvertantly. If this occurs, results are unpredictable.

For a task to use these resources, you must set up the task's TELN in your TSK.X routine. You cannot change a task's TELN after the task has been initiated.

Providing Even More Resources

If a task needs resources in addition to floating-point hardware and contiguous memory locations, you can write your own ESV.X routine to handle the extra resources and use the supplied handler as a subroutine. From within your own ESV.X routine, call out to the supplied handler, using the alias ESV.A instead of ESV.X, with the accumulators set up appropriately.

Operator Communications

When you issue a .QTSK task call, you must pass in AC2 the address of a task queue table, whose format and length are as described in Chapter 5. When the scheduler calls out to TSK.X, it passes the queue table address in AC1. Thus, you can append additional information to the queue table (that is, you can supply a longer table), and access this information from within TSK.X.

The operator communications feature (OPCOM) describes "programs" to be run from the console by means of a "program table" consisting of "program frames" of a given length. When the operator types a QUE command, the scheduler copies information from a program frame into a queue table. You can increase the size of a program frame and thereby have the scheduler pass additional information about a program to TSK.X via a longer queue table.

To do this, define symbol LPN.X with .ENT and set it equal to the number of additional words in each program frame. On an OPCOM QUE command, these words will be copied, in order, to the end of the associated task queue table, where they will be accessible to TSK.X.

Task Control Block Values

Table J-1 describes the initial values the scheduler assigns to words in a TCB and when these values can be changed during a task's lifetime. A bracketed number indicates a note. Entry *Name* is the symbol in PARU.SR which represents the offset within the TCB. *Initial Contents* describes the value placed there by the scheduler and seen on input to TSK.X. In column *.TASK?* a "Yes" means that TSK.X can set or change the contents of this word if .TASK is initiating the task; "No" means that TSK.X can't change this word. A "Yes" or "No" in column *.QTSK* means the same thing for .QTSK. In column *Later?* , a "Yes" means that this word can be changed later in the task's life; "No" means it cannot be changed. A bracketed number indicates a note. In the *Initial Contents* column, the entry applies to both .TASK and .QTSK, unless there are two entries separated by a slash (/); in this case, the first entry applies to .TASK and the second to .QTSK.

Table J-1. TCB Words and How They Can Be Changed

Name	Initial Contents	.TASK?	.QTSK?	Later?
TPC	B0-14:Start addr;B15:Undefined	Yes	Yes	Yes
TAC0	Undefined / System-maintained	Yes	No	Yes
TAC1	Undefined / System-maintained	Yes	No	Yes
TAC2	AC2 at .TASK/System-maintained	Yes	No	Yes
TAC3	K.ILL [1]/System-maintained [2]	Yes	No	Yes
TPRST	B0-7:0 ; B8-15: Start pri.	Yes	Yes	[3] [4]
TSYS	System-maintained	No	No	No
TLNK	System-maintained	No	No	No
TUSP	Undefined	Yes	Yes	[5]
TELN	0	Yes	Yes	[6]
TID	Task identifier	No	No	No
TTMP	System-maintained	No	No	No
TKLAD	0	Yes	Yes	Yes
TSP	Undefined	Yes	Yes	[5]
TFP	Undefined	Yes	Yes	[5]
TSL	Undefined	Yes	Yes	[5]
TSO	Undefined	Yes	Yes	[5]

Notes:

1. Address K.ILL is the entry for the .KILL task call code. This address is placed in TAC3 so that a task can kill itself by simply returning to the address it receives in AC3.

2. At TSK.X time for a task initiated by Q.TSK, TAC3 does not contain the address K.ILL. However, after TSK.X completes, but before the new task gains control, the scheduler places the address K.ILL in TAC3, so that the task's initial AC3 wil be correct (see also note 1).

3. The modification of a task's status bits must be an indivisible operation. The interrupt world can modify the status bits on suspended tasks only. Thus, modifying the status bits of a ready task must be a "task-indivisible" operation, while modification of a suspended task must be an "interrupt-indivisible" operation. Scheduler state provides task-indivisibility, as does interrupt-disabled state. Interrupt-disabled state, and, on an ECLIPSE machine, the use of bit instructions provides interrupt-indivisibility.

4. Don't modify the priority portion of TPRST; use the task call .PRI instead.

5. Because these values are saved and restored only on task swaps (not by .TSAVE, as are the accumulators, for example), it is meaningless to change the values while in scheduler state. Instead, you should change the actual storage locations. Change USP (16_8) instead of TUSP. On an ECLIPSE computer, change locations 40 through 43_8 instead of TSP through TSO. On a NOVA computer with a hardware stack, change the hardware stack and frame pointers and locations 42_8 (stack limit) and 46_8 (instruction trap PC).

6. As mentioned earlier, you cannot change word TELN after TSK.X time, if you use the additional resource handler (ESV.X routine) supplied in SYS.LB.

End of Appendix

Index

Within this index, the letter "f" following a page number means "and the following page"; "ff" means "and the following pages". Entries that begin with a period and have no type description (e.g. .APPEND) are RDOS system or task calls. Entries are alphabetized as if they were all lowercase without periods.