

DATA GENERAL  
CORPORATION

Southboro,  
Massachusetts 01772  
(617) 485-9100

INTRODUCTION  
TO THE  
REAL TIME DISK OPERATING SYSTEM

ABSTRACT

This document provides an introduction to the DGC Real Time Disk Operating System (RDOS). It also contains a basic description of the concepts and terms found in other documentation describing the use and structure of RDOS.

This revision of the Introduction to the Real Time Disk Operating System , 093-000083-02, is a major revision and supersedes manual number 093-000083-01. For a description of changes made in this revision, see the list of changes at the end of this manual.

## TABLE OF CONTENTS

An Introduction to RDOS .....	1
RDOS Organization .....	3
System Generation .....	5
Foreground - Background Programming .....	7
Foreground - Background with Memory Protection .....	9
Dual-Processor and Shared-Disk Systems .....	11
Communication with RDOS .....	13
Command Line Interpreter (CLI) .....	15
Program Swaps and Chains .....	17
User Program Segmentation .....	19
Tasks .....	21
Task States and Priorities .....	23
Task Environments .....	25
Task Execution Control .....	27
Intertask Communication/Synchronization .....	29
Task Timing Control .....	29
Real Time FORTRAN .....	31
System Calls .....	33
RDOS Input-Output Control .....	35
Input/Output Command Modes .....	37
System Input/Output Buffering .....	39
Spooling .....	39
Buffer Control Package .....	39
Disk File Organization .....	41
Disk File Structures .....	43
Disk File Input/Output Control .....	45
Interrupt Servicing Program .....	47
User Interrupt Processing .....	47
Multiple Devices and Units .....	49
System Library .....	51
RDOS Supported Software .....	53

## AN INTRODUCTION TO RDOS

Data General's RDOS is a Real Time Disk Operating System. RDOS is real time oriented since it can schedule and allocate program control to many different subprogram tasks to provide simultaneous disk operation, to maximize throughput, and to insure efficiency and economy of operation.

Data General's RDOS resulted from the desire to provide a real time system that had the capabilities of both the existing Disk Operating System (DOS) and the core-only multi-tasking Real Time Operating System (RTOS), a compatible subset of RDOS. Feedback from customers and potential users of the Nova family of computers also indicated some additional features that would be desirable in a new system.

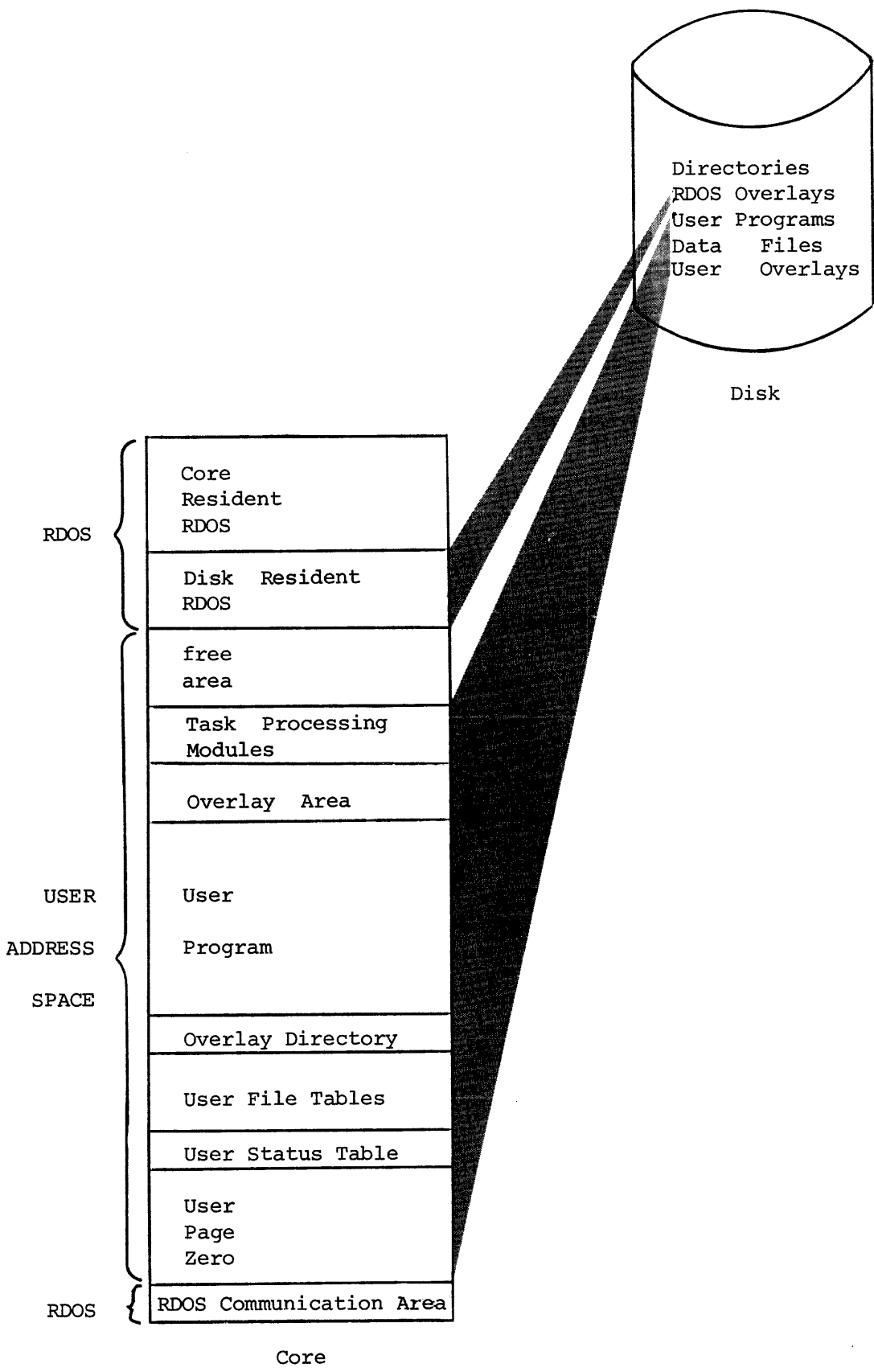
A modern real time operating system must be geared to change and diversity. The RDOS system itself can exist in an almost unlimited variety of machine configurations: different installations will typically have different configurations as well as different applications. Moreover, the configuration and scope of the project at a given installation may frequently change. We see that the operating system must cope with an unprecedented number of environments. All of this puts a premium on the system modularity and flexibility which has been designed into RDOS.

To obtain the full capabilities of the Real Time Disk Operating System, the user only requires a Nova computer with 12K words or core memory, a real time clock, console Teletype, and a disk. In addition to this minimum machine configuration, RDOS supports additional core storage (in excess of 65K words), 2 million words of fixed head disk, four disk cartridges or pack drives, eight magnetic tape transports (7 and/or 9 track), card readers, line printers, communication equipment, analog and digital front end equipment.

Some of the major features of RDOS are:

- . Operating system not completely core resident
- . Modular multi-task monitor
- . Multiple user overlay areas
- . 256 software levels of task priority
- . Spooling (disk buffering) of output
- . Flexible disk file structures
- . Buffered and non-buffered I/O
- . Support for Real Time Fortran IV
- . Hardware protected foreground/background programs

The modular structure of the RDOS multi-task monitor permits it to be tailored by the user at program load time to include only those real time features which he currently needs. This tailoring promotes more efficient core utilization for each single or multi-task user program supported by RDOS.



RDOS Organization

## RDOS ORGANIZATION

The RDOS executive constitutes the main framework of the operating system, and must be resident in permanent core storage before any continuous and coordinated processing can take place. Functions performed by this resident portion of RDOS include interrupt processing, overlay and buffer management, system call processing and the device interrupt servicing routines. Other modules of the system are brought into core from disk storage as they are required to perform specific functions such as full or partial system initializations, file maintenance operations like opening, closing, renaming, or deleting a file and spooling control.

The RDOS system occupies two areas in memory. The lowest 16g memory locations are used for entry points (interrupt and program) into the second area located at the top of memory. The lowest address used by this second module is defined as one greater than the highest user memory address available (HMA). The value of HMA is determined by system generation and is a function of the user's application and system hardware configuration.

The portion of page zero memory available for user programming begins at location 16 (labeled USP), and extends through location 377. USP is a special location preserved by RDOS whenever program control goes from one task to another.

There is one User Status Table for each program level. This table extends from 400 octal through 427 and contains information describing the user program such as its length, and the number of tasks and active I/O channels that the program specifies. Above the UST are individual User File Tables, one for each channel in the user program. The UST contains the name of the file associated with the channel and the file attributes.

Above the last User File Table is an area reserved for the pool of TCBs. This pool contains both the active and the inactive TCB chain. If the user overlays are called in the program, an overlay directory will be found above the TCB pool. Above the overlay directory lies the user program.

After loading the user program, the relocatable loader will load all modules referenced by the user program. These modules will be extracted from libraries like the System Library. By default, System Library modules will be the last to be loaded, after the user program and overlay areas. In this group will be the Task Scheduler and other task monitor modules.

SYSGEN  
CORE STORAGE (IN THOUSANDS OF WORDS) 20  
ENTER NUMBER OF STACKS (1-5) 5  
RESPOND "1" (YES) OR "0" (NO) REGARDING SYSTEM CONFIGURATION  
RTC? 1  
ENTER RTC FREQ (1=10HZ,2=100HZ,3=1000HZ) 1  
DSK? 1  
ENTER DISK STORAGE (IN THOUSANDS OF WORDS) 256  
DKP? 1  
ENTER NUMBER OF DEVICES 2  
ENTER NUMBER OF SECTORS/TRACK 12  
ENTER NUMBER OF HEADS 2  
ENTER MASTER DEVICE DSK  
ENTER BOOTSTRAP DEVICE DSK  
MTA? 1  
ENTER NUMBER OF DEVICES 2  
PTR? 1  
ENTER NUMBER OF DEVICES 1  
PTP? 1  
ENTER NUMBER OF DEVICES 1  
LPT? 1  
ENTER COLUMN SIZE 80  
ENTER NUMBER OF DEVICES 1  
CDR? 1  
PLT? 1  
QTY? 0  
SECOND TTY? 1

R

System Generation

## SYSTEM GENERATION

In real time operating systems, individual user installation requirements may vary from installation either in the hardware itself or in dissimilarities inherent in the application. These differences may take the form of different applications, different configurations of standard hardware, special process input/output hardware, core storage sizes, system throughput and priority considerations.

This means that each installation must be tailored to the specific system function requirements and input/output configuration of that installation. The tailoring function is defined as system generation (SYSGEN). SYSGEN provides facilities for the creation of a monitor system composed of DGC and user written programs and subroutines. The end product of system generation is a disk resident operating system which is custom built to provide an efficient executive system for a specific machine environment.

In the DGC Real Time Disk Operating System, the builder of the tailored operating system is an executable program that can operate on any RDOS configuration. SYSGEN permits a system to be constructed for one or more fixed head disks, disk cartridges, or disk packs from relocatable program modules stored in library files. Furthermore, the user installation may modify the DGC - supplied configuration, deleting those functions not required by the installation and adding installation-created functions and programs.

The modular design and availability of numerous features and attachable units make possible multiple RDOS configurations tailored to individual application requirements.

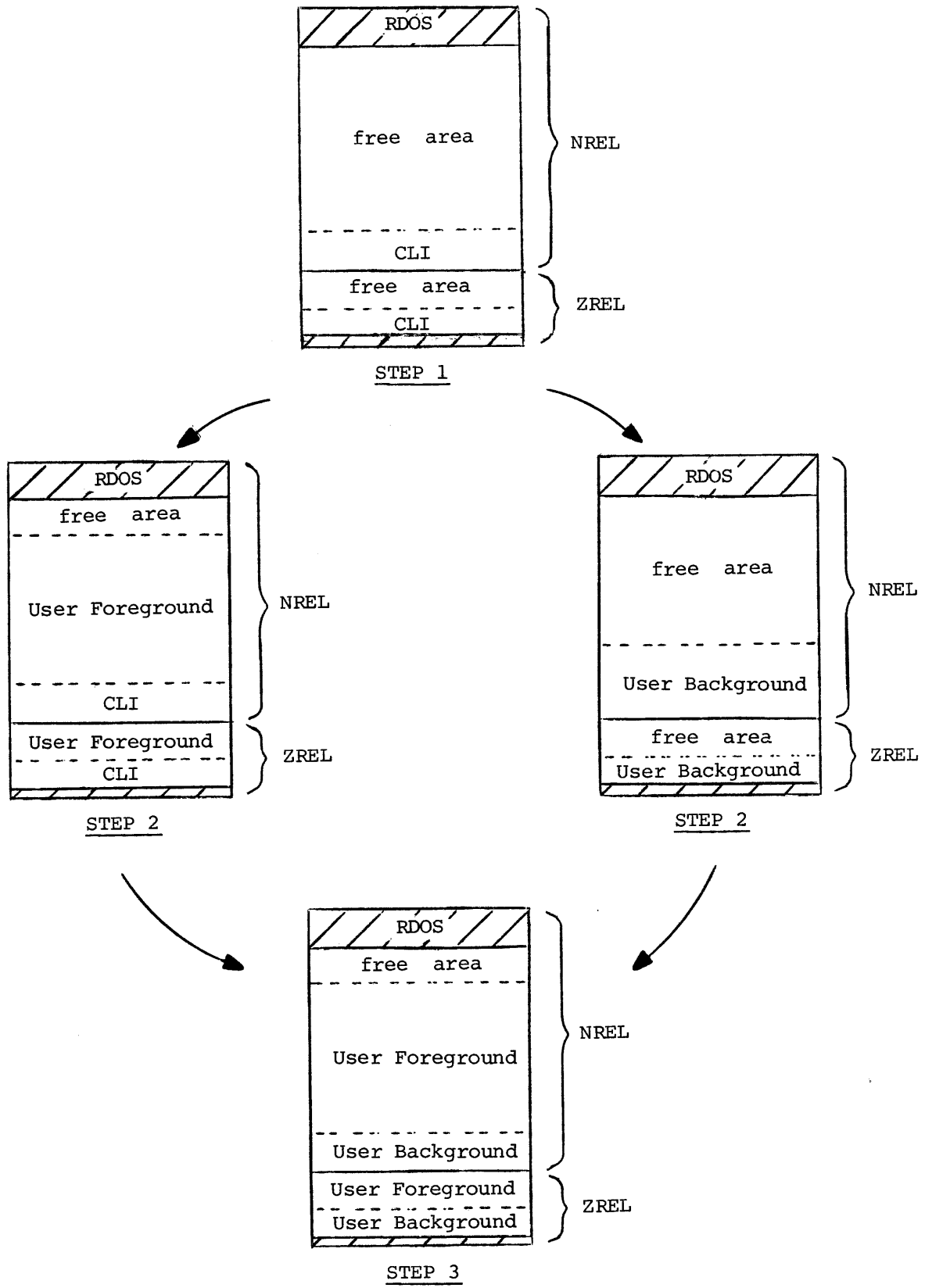
To assist users in generating their initial system, a standardized starter system called a BOOTSTRAP system is provided with each customer installation, and contains the basic elements essential for system generation in a form that will be directly usable by a majority of customers. It is designed to support a minimum hardware configuration:

- Nova Computer
- 12K words of memory
- Console Teletype
- Fixed or moving head disk

When they are available, it will take advantage of additional devices such as:

- Paper tape reader
- Paper tape punch
- Line printer
- Magnetic tape





Loading Foreground and Background Programs

## FOREGROUND - BACKGROUND PROGRAMMING

In most current computer installations, real time control and program development functions are performed sequentially. That is, program development must be completed before any real time program may be run. Moreover, once a real time program is run, no other programs can be run concurrently even though the real time program itself requires only a fraction of available system resources at any given moment.

To increase system utilization, RDOS permits multiprogramming. Multiprogramming permits unrelated collections of tasks to be performed concurrently, sharing basic system resources. Thus the total system is kept as productive as possible continuously. Two separate programs may be run under the RDOS multiprogramming system: a foreground and a background program. Priority for CPU processing time between these two programs is based on a multi-level software/hardware interrupt hierarchy controlled by RDOS. All foreground program tasks have priority over all background tasks.

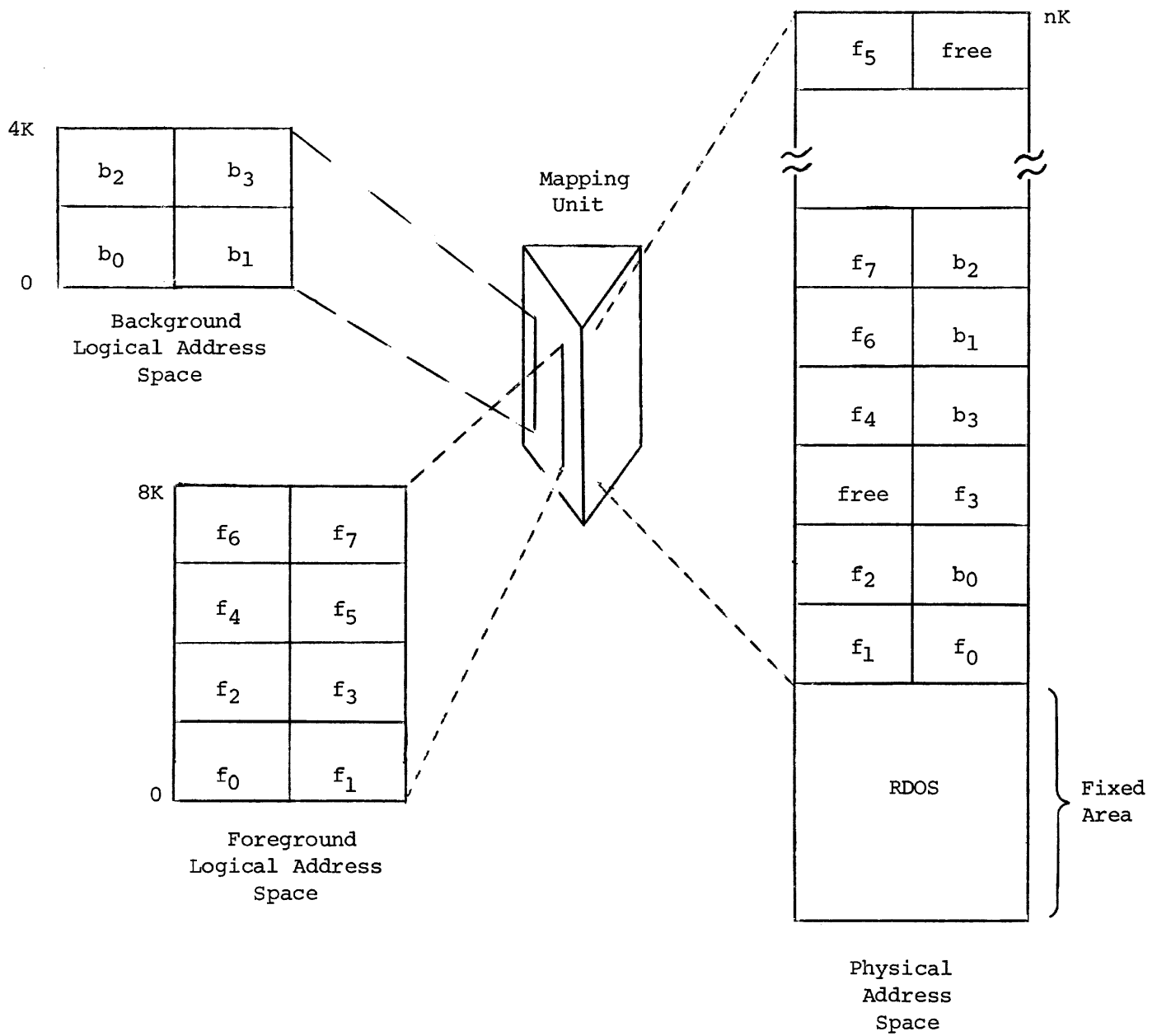
A typical foreground - background system might contain a real-time process control program in the foreground area and an assembly, compilation, or payroll program at different times in the background area.

Separating the foreground and background programs in core are software partitions created during the relocatable loading of the two programs. There is both a page zero (ZREL) partition and a normally relocatable (NREL) partition. Each indicates the starting address of the foreground program area and is indicated by dashed lines in the illustration on the preceding page. Programs may access common disk files, yet disk file integrity remains the responsibility of the user.

The illustration on the preceding page outlines three steps taken to load and run a foreground - background system. Step 1 illustrates a typical core map after a disk bootstrap has been performed. The Command Line Interpreter alone is resident in user address space.

In step 2, one of two procedures may be followed. First, the CLI may be used to load a previously developed user foreground program by means of the EXFG command, which loads a foreground program into memory and transfers control to it. However, EXFG will load a foreground program only if it can coexist with the CLI. The foreground partitions were set when the program was loaded by means of the relocatable loader. Alternatively, the CLI may be used in step 2 to load another background program, overwriting the CLI and pushing to a lower program level.

To make the transition from step 2 to step 3, a full foreground - background environment, one of two procedures must be followed. Either the background program must be loaded by means of the CLI, or the foreground program must be loaded via an .EXFG system call issued from the background program. The CLI may be restored in memory by having the user background program issue a .RTN . Similarly, the foreground program may be terminated by issuing a .RTN . Both the foreground and background programs may be individually terminated by issuing a keyboard interrupt.



Logical to Physical Address Mapping

## FOREGROUND/BACKGROUND WITH MEMORY PROTECTION

A memory mapping and protection device has been developed for the NOVA 800 series computers which will be supported under the Real Time Disk Operating System.

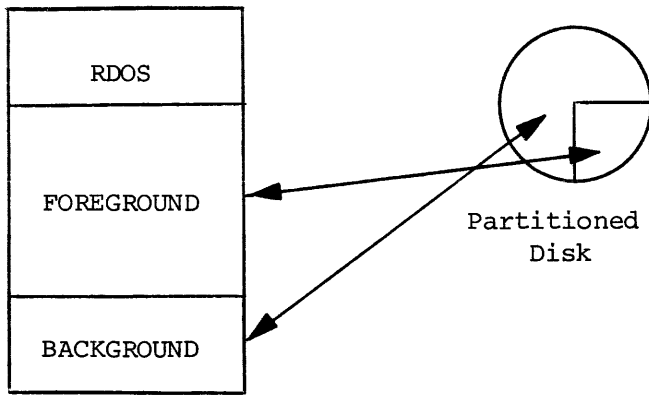
This device extends the maximum core configuration for a single CPU from 32K to 128K. Within the framework of an executing program, two modes exist. The first mode is the absolute mode. In this mode, only the lower 32K is directly addressable and the mapping device is not used. RDOS resides in the low physical memory locations and executes in the absolute mode.

The second mode is called the mapped or user mode. In this mode, up to thirty-two  $1024_{10}$  word blocks of memory are mapped using the device so to appear as a logical 32K address space to the executing program. The foreground and background programs execute in user mode and are not aware of their actual memory locations.

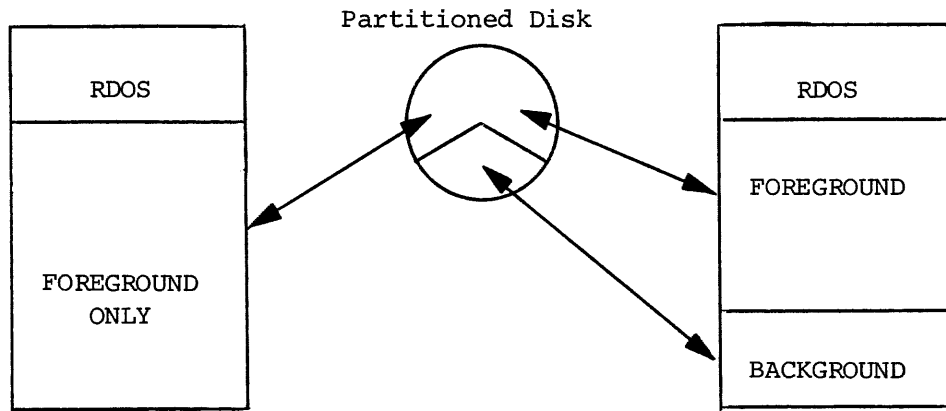
Any program operating in user mode uses a complete logical address space including its private page zero and extending through its upper memory bound (NMAX). NMAX is determined by the requirements of the individual program in  $1024_{10}$  word increments, and may extend as high as 32K. The operating system is responsible for assigning free memory from its available pool to the user program prior to its execution. The technique used to manage the mapping device and the construction of the user program in a logical address space is also the responsibility of RDOS.

While a user program is running, it may communicate with the operating system via the standard RDOS .SYSTM commands (including those provided for managing user-defined I/O devices).

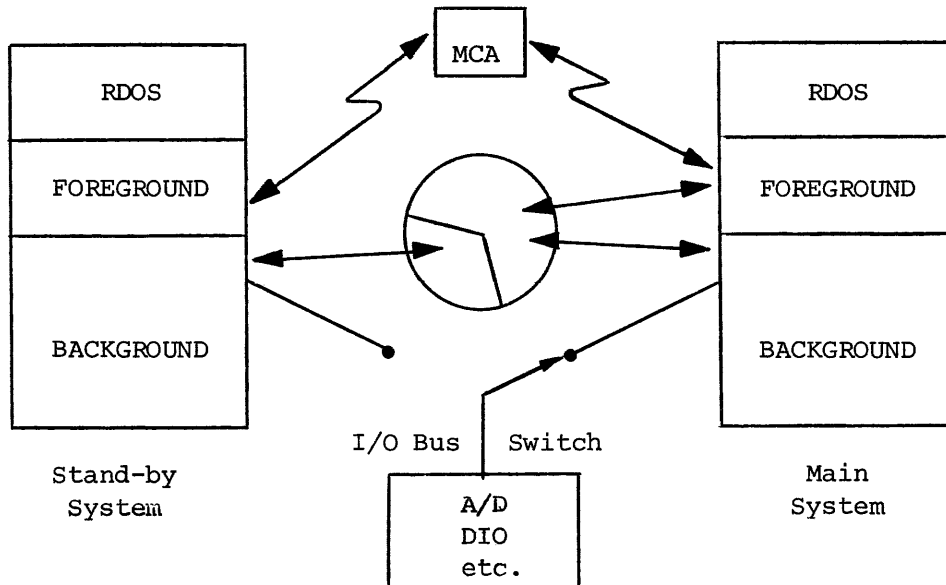
A user defined device may not utilize the data channel. The data channel is managed via a second mapping device, and consequently its utilization must be allocated by the operating system.



Single Processor / Partitioned Disk System



Dual Processor / Shared Disk System



Parallel Main and Standby System

## DUAL-PROCESSOR AND SHARED-DISK SYSTEMS

As the cost of small computers continues to drop, many new application areas are developing for multi-processor systems. Such systems commonly consist of two CPUs which share some or all of the system peripherals, thus lowering the total cost of the systems. Multi-processor systems permit processors to communicate via disk files on a common disk. Portions of disk space are also protected or partitioned to preserve each processor's disk file integrity. (Such disk file segmentation can also be useful in single-processor systems.)

Dual-processor systems contain a main system and a back-up system. The main system controls or monitors some process or series of processes continuously. The back-up system stands ready to assume the main system functions in the event of failure in the main system. While in the standby state, the back-up system can be employed on lower priority tasks such as data analysis, summary reporting, and the development of new real-time programs. Both the main and the back-up systems can run under the Real Time Disk Operating System, either in a foreground/background arrangement or as a single real time program.

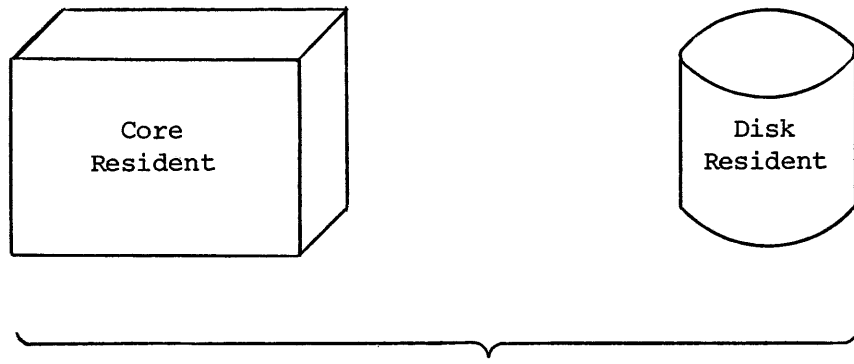
Some method of communicating information between the main and the back-up systems must be continuously maintained. Often the means is a CPU interconnection link treated as an I/O device, or a bulk storage medium such as a disk or tape that both CPUs can access. The communication must be operated continuously or at some periodic rate compatible with the particular system's requirements for smooth transfer of control.

If the main system fails, some method for detecting the trouble must be available. Such methods include extensive software cross checking of critical functions and the use of such special hardware as a watchdog timer. If failure occurs, transfer to the back-up system must be made quickly with a minimum disturbance of the controlled process. To provide a smooth transfer of control, process information can be passed via disk files, via a multiprocessor communications adapter (MCA), or by means of some other hardware interface between the two systems.

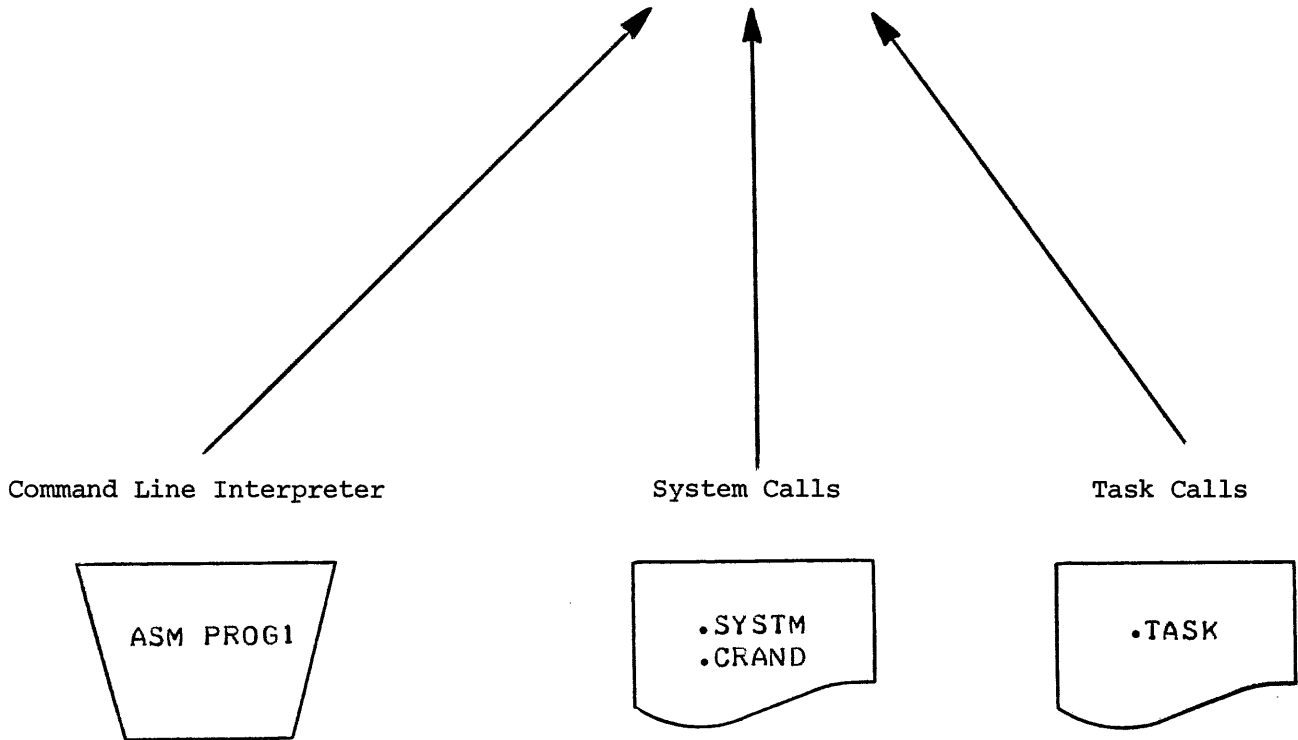
To permit several program areas to use a common disk independently, that disk's storage must be partitioned. Partitioning must also be done if complete disk file integrity is to be assured in single processor foreground/background systems.

Disk partitions are subsets of the total file space on a disk; partitions are established by the user via system calls or the CLI. Each partition contains its own file directory and its own bit allocation map (MAP.DR). There are two types of partitions: primary and secondary. A primary partition includes all of a disk's file space, while the secondary partition's file space is a subset of the primary partition.

Files in the primary partition may be accessed (linked) by file directories in the secondary partitions, although files in the secondary cannot be accessed directly by entries in the primary directory. Distinct file names may be created within each partition and appear in the partition directory.



REAL TIME DISK OPERATING SYSTEM



Communicating with RDOS

## COMMUNICATING WITH RDOS

There are three principal ways for a user to interface with RDOS and to make the system work for him. These ways are (1) with system calls, (2) with task calls, and (3) via the Command Line Interpreter (CLI).

System calls and task calls are issued as program instructions, while the CLI is a dynamic interface to RDOS via the teletype console. System calls and task calls activate logic within either the system or task modules.

The Command Line Interpreter (CLI) is a system program that accepts command lines from the console teletypewriter and translates the input as commands to the operating system. The CLI is basically a string handler that acts as an interface between the user at the teletypewriter and the system. In addition, the CLI performs certain file housekeeping chores for the user.

The system restores the CLI to core whenever the system is idle -- after initialization, after a bootstrap, after a teletype break, after execution of a program, etc.

The CLI indicates to the user that the system is idle and the CLI is ready to accept commands by typing a ready message on the teletypewriter. The message consists of "R" followed by a carriage return.

The user activates CLI responses to a command by typing a line and pressing the RETURN key or the CTRL L (form feed) keys. The CLI will not respond until RETURN or CTRL L is pressed.

In addition to the above means of communicating with RDOS, operators obtain the system's immediate attention by issuing one of three console keyboard interrupts: CTRL A, CTRL C, and CTRL F.

CTRL A interrupts the currently executing user program (or the background program in a dual program environment) and returns control to the CLI. This interrupt is a simple abort, giving no means of continuing the program at the point where it was interrupted. CTRL C, by contrast, interrupts the currently executing program as does CTRL A yet preserves a snapshot of this program: file BREAK.SV . Thus CTRL C allows an interrupted program to be restarted.

CTRL F is issued to abort the foreground program and, like CTRL A, returns control to the CLI.



## CLI Commands

ALGOL - Compile an ALGOL source file.  
APPEND - Append one, two or more files to produce a single file.  
ASM - Assemble a program.  
BLDR - Load an absolute binary tape with the binary loader.  
BPUNCH - Punch a file in binary on the high speed punch.  
CCONT - Create a contiguous file.  
CHATR - Change the attributes of an existing file.  
CLG - Compile, load, and execute FORTRAN programs.  
CREATE - Create a file or series of files.  
DEB - Read in a program from disk and transfer control to the debugger.  
  
DELETE - Delete a file or a series of files.  
DIR - Change the current default directory device .  
DISK - Obtain a count of the number of blocks used and the number of blocks available on the default device.  
  
DUMP - Dump files. The dump includes directory information for each file which enables their later reloading.  
EDIT - Edit or build source files in the background.  
EXFG - Execute a program in the foreground.  
FEDIT - Edit or build source files in the foreground.  
FILCOM - Compare two files, word by word.  
FORT - Compile and assemble a FORTRAN source file.  
GTOD - Get the time and date.  
INIT - Initialize a directory device or magnetic tape transport.  
INSTALL - Specify a save file for use in bootstrapping.  
LFE - Update RDOS library files.  
LIST - List names of files in the default file directory with their length in bytes and their attributes.  
  
LOAD - Reload dumped files.  
MAC - Perform a macro assembly.  
MKABS - Make an absolute binary file from a saved file.  
MKSAVE - Make a save file from an absolute binary file.  
OEDIT - Examine or modify locations' contents in octal.  
PRINT - Print a file on the line printer.  
PUNCH - Copy an ASCII file on the high speed punch.  
RELEASE - Prevent further I/O access to a directory device, or rewind the magnetic tape.  
  
RENAME - Change the name of a file.  
RLDR - Load a save file from a series of relocatable binary files.  
SAVE - Name a save file created by a CTRL C interrupt.  
SPDIS - Disable spooling.  
SPEBL - Enable spooling.  
SPKILL - Stop a spool operation.  
STOD - Set the time and date.  
TYPE - Output an ASCII file on the TTY printer.  
XFER - Transfer the contents of one file (or device) to another file (or device).

## COMMAND LINE INTERPRETER (CLI)

Having gained control, the CLI can perform a variety of necessary functions for the operator sitting at the console terminal.

The CLI itself executes certain system commands such as CREATE and RENAME. More complex commands cause the CLI to build a file containing an edited version of the command line and then to load the program named in the command line for execution. When execution is finished, control is returned to the CLI.

Some of the functions performed by the CLI are:

- . System Initialization and Installation
- . File Creation and Maintenance
- . File Display and Reproduction
- . Setting/Reading the Time of Day
- . Spooling Control
- . Interfacing to System Software

CLI commands can be stacked (new ones can be issued while the current command is still being performed), and a variety of symbol conventions can be added to the basic commands to extend their meaning. Among these symbols are global switches which are appended to CLI commands themselves and local switches which are appended to CLI command arguments.

A list of all CLI commands is given on the previous page. These commands range from the simple to the complex. A simple command

```
DISK ↓
```

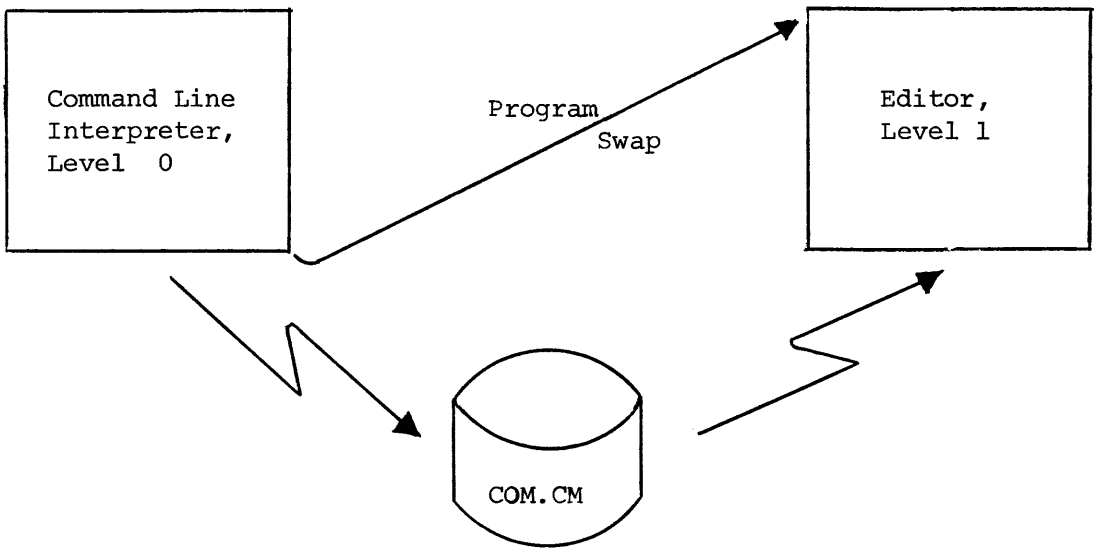
causes a count to be printed of disk blocks used and disk blocks still available on the default directory device. A complex command like

```
RLDR/D MAIN TIMPLT QUAD FMT.LB FORT.LB $LPT/L ↓
```

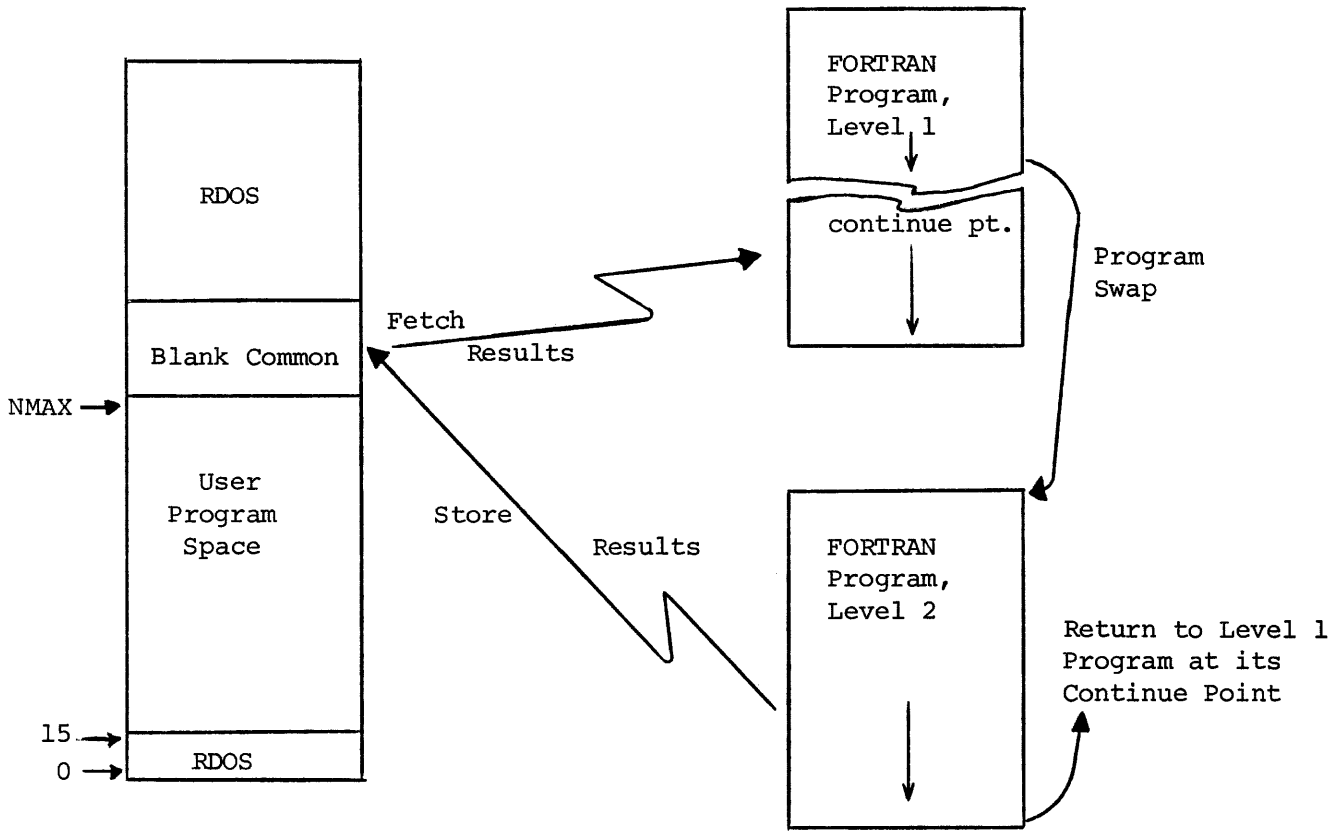
causes a real time FORTRAN program to be loaded with the debugger, and a load map to be printed on the line printer.

If commands issued to the CLI cannot be executed, the CLI responds with appropriate error messages. Commands requiring operator intervention (like mounting paper tape in a paper tape reader) cause prompting messages to be given. The spooling feature (to be discussed later) permits the CLI to execute new commands while it is waiting for previous I/O operations to be completed. Thus, program editing could be continued while an assembly listing is still being output.

To allow a user to take full advantage of the CLI and its command repertoire, and to permit the operation of the CLI within 12K core configuration, the CLI program has been subdivided into smaller portions called program overlays.



Communicating Via a Disk File



Communicating Via Blank Common

## PROGRAM SWAPS AND CHAINS

Any program running under RDOS can suspend its own execution and either invoke another distinct program or call for a new section of itself. The current program is overwritten in both cases.

Every new program (or section of the same program ) which is invoked must exist as a disk save file. If the file constitutes an entirely new program, it is called a program swap. If the file is merely a new section of the current program, the file is called a program chain. Both program swaps and chains contain core images extending from address 16 (USP) through the highest address in the user program (NMAX). Not included in this area is blank common (in the case of FORTRAN programs).

The program whose core images are overwritten is stored temporarily on disk, along with its User Status Table. Information in this table enables the program to be restarted upon termination of a swap. Each entire program is considered to reside at a program level. The Command Line Interpreter (CLI) is just such a program, and it resides at level 0 (the highest level in the system). Correspondingly, a program swapped into core resides at a level which is one lower than that of the caller. Program chains, segments of the same program, reside at the same level.

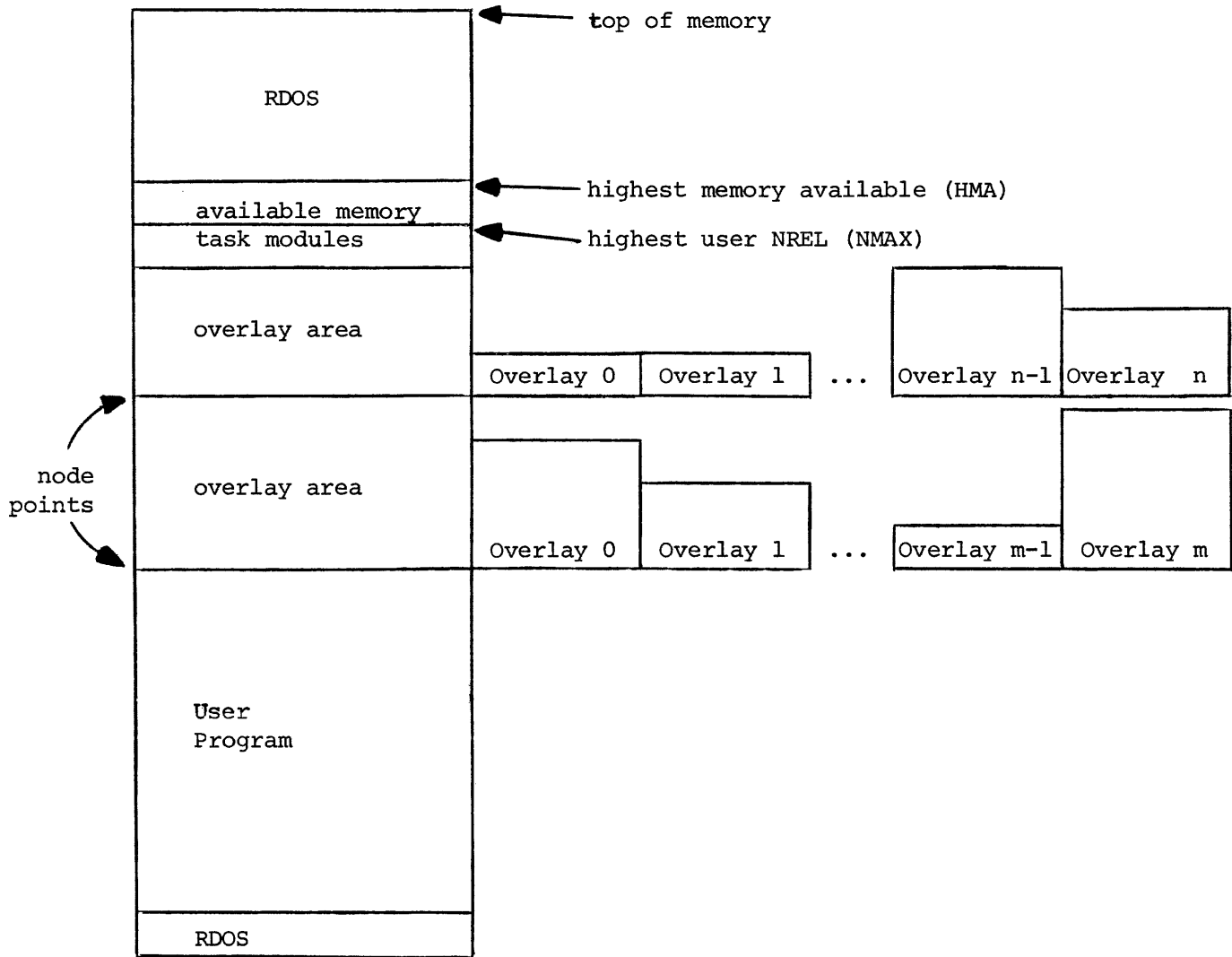
The program chain facility permits programs to be run which require more core storage than is ever available at one time. To use chaining, a program must be written in serially executable segments, each of which calls the next segment.

The program swap facility, by contrast, permits distinct programs to call one another in much the same manner as subroutines are called. The primary difference is the size of the routine and the manner of argument passing. Whereas subroutines are always smaller than the total amount of resident core, program swaps are as large as total user program space (or as large as multiples of user address space if chaining is performed).

The operating system permits up to five levels of program swaps to occur. This implies that a program called by the CLI can in turn invoke a third, etc. The nesting of program swaps can therefore occur to a level of four distinct swaps (with a virtually unlimited number of chains for each program level).

There are two ways that parameters can be passed to program swaps and that results can be returned to the callers. These means are: (1) via disk files with agreed upon names, and (2) via blank common. Communication via disk files is possible under both program swaps and program chains. The CLI uses this method in passing switch and file name information to utility programs which it then calls; the name of the disk file that is written by the CLI and read by utilities is COM.CM .

FORTRAN program swaps may pass arguments via blank common provided they observe two requirements. These requirements are that all participating FORTRAN programs be either single tasking or multitasking (not both), and the size of blank common must be the same for each.



RDOS Memory Map With Overlays

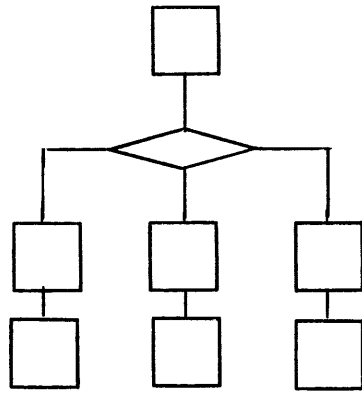
## USER PROGRAM SEGMENTATION

When the user address space of an RDOS system is not sufficient to hold all the necessary programs at one time, some form of program read-in scheme must be employed whereby programs, upon demand, are brought in from disk storage. The RDOS system reserves part of the user address space for this program read-in, and divides it into fixed-length partitioned core storage areas which form a repository for programs of a limited size. This allows the RDOS user to make a segmentation of his program into one or more parts which fit into the fixed-size core areas at execution time. These program segments are called user overlays and are stored on disk in core image format to facilitate rapid loading when execution is desired.

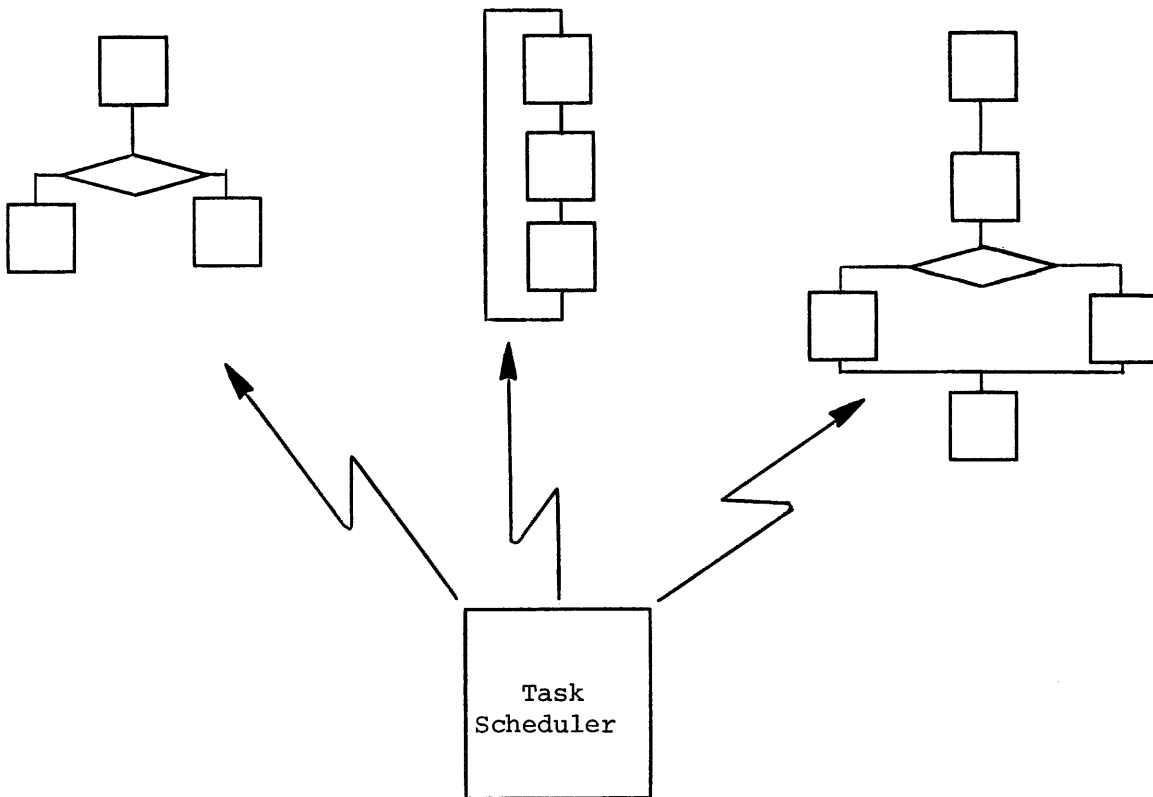
User overlays differ from program swaps in that user overlays overwrite only a fixed area within a root program which remains active and core resident during the load time. RDOS itself uses overlays to enhance its own operation; these are called system overlays to distinguish them from user overlays. The complete set of user overlays used by a root program is called a user overlay file.

The beginning of each overlay area is called its node point. Associated with each area is a collection of one or more user overlays. Each overlay area within the root program must be large enough to accommodate the largest overlay associated with it. There may be up to 8 overlay areas in each root program.

User overlays can be called in either single or multi-task environments. Overlays are created at the same time the executable root program is built by the relocatable loader.



Single Task Environment



Multi-task Environment

## TASKS

A task is a logically complete active execution path through a program, subprogram, or overlay which demands use of system resources (usually CPU control). Single task environments are already familiar to users of non-real time operating systems. FORTRAN IV programs, most user assembly level programs, even system utility programs like the assembler, editor, etc., are all examples of programs running in a single task environment. A single task environment is a program which has a single unified path connecting all its program logic, no matter how complex the logic branches.

There is no compelling reason why a program should be limited to performing just one task, however. Indeed, most user environments contain a multitude of unrelated tasks which must be performed. It was the problem of efficiently controlling real time environments which led to the notion of multi-task real time operating systems.

One real time program may have from several to a virtually unlimited number of logically distinct tasks. Each task performs a specified function asynchronously and in real time (i.e., as close to instantly as possible). CPU control is allocated by the RDOS Task Scheduler to the highest priority task that is ready to perform or continue performing its function.

Examples of multitask environments are airline reservation systems, inventory control systems, process control operations, and communications networks with message queuing and switching. The individual tasks within an inventory control system might be the immediate updating of inventory totals from data received by remote terminals, and the reordering of those items whose quantities fall below specified reorder points.



## Task States

- EXECUTING - A Task has control of the Central Processing Unit.
- READY - A Task is available for execution, but is waiting for a higher priority Task to be readied, suspended, or killed.
- SUSPENDED - A Task is awaiting the occurrence or completion of some system call or real-time operation.
- DORMANT - A Task has not been initiated in the system or its execution was completed.

## TASK STATES AND PRIORITIES

User multi-task programs run under RDOS will have one task (called the "default task") initiated for them by default. If the user wants more tasks, he initiates these tasks by issuing the appropriate task monitor call from this first task.

When a multi-task environment is established there becomes a requirement for a task scheduler to decide which task should be executing. To enable the scheduler to function, each task is assigned a priority at the time of initiation. Within RDOS there can be 256 levels of task priority in the range 0 to 255, with priority 0 being the highest. Several tasks may exist at the same priority level.

The default task is initiated at the highest priority and since it is the only task in the system it receives control. When this task initiates the second and subsequent tasks, the task scheduler is called upon to put the highest priority task into execution. Other tasks that have been initiated but are of lower priority are said to be ready to run.

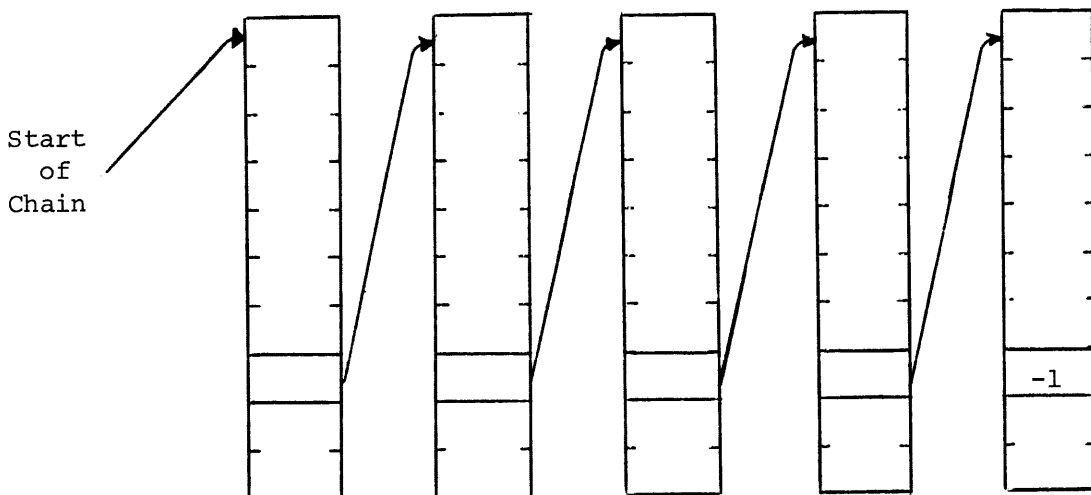
The executing task becomes suspended when it makes a call to the operating system to perform some function such as an I/O transfer, get time of day, etc. The task remains suspended until the operation is completed, at which time it becomes readied.

Thus we have seen that tasks under the RDOS system can exist in either of four states. Tasks are in control of the CPU and are executing their assigned instruction paths, they are ready and waiting for their turn to gain control, they are suspended and awaiting the occurrence or completion of some real-time operation, or they are dormant and have not been initiated into one of the other states.

Task Control Block Structure

word 0	User PC and Carry	Used to store the task's Program Counter and Carry and Carry.
1	AC0	Used to store the task's AC0.
2	AC1	Used to store the task's AC1.
3	AC2	Used to store the task's AC2.
4	AC3	Used to store the task's AC3.
5	Status bits and Priority	Contains task priority and task status.
6	System call word	Used by RDOS when task issues a system call.
7	Link	Points to the next TCB in the chain.
10	USP	Contents of location 16, used for general purpose storage or for FORTRAN STACK POINTER.
11	TELN	Points to the Run Time Stack segment associated with this task if it is a FORTRAN task. (Otherwise, unused.)

TCB Chain



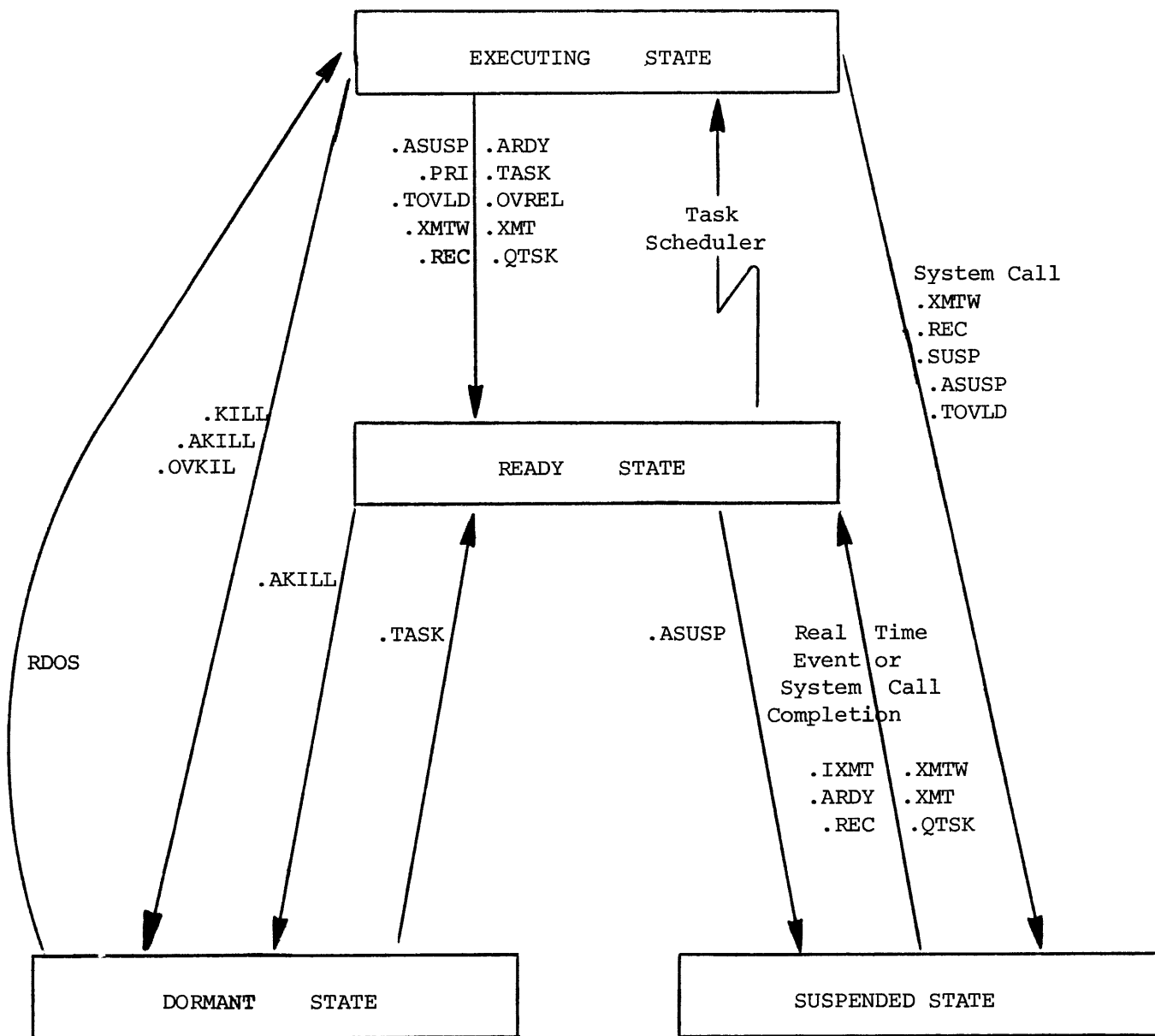
## TASK ENVIRONMENTS

As discussed in the previous section, a task within an RDOS system can either be dormant or active. If the task is dormant, the system does not know that it exists even though the code could be resident within the user address space. When a task is active (executing, ready or suspended state) certain status information must be maintained about each task to enable the scheduler to maintain the highest priority ready task in the executing state.

This status information about each active task is contained within an information structure called a Task Control Block (TCB). There is one TCB for each task in the active state (and no TCB for a dormant task). TCBs are used to store active register states and other priority and status information when the task exists in either the ready or suspended state. The TCB of the executing task is allocating to the task but the TCB remains unused by the system until the task loses control of the CPU. If the task becomes readied or suspended, its TCB is then used to store its status information. If, on the other hand, the rescheduling resulted from the task terminating its own execution, its TCB is placed into a pool of available TCBs.

The TCBs of ready, executing, and suspended tasks are linked together in an active chain. This chain is organized in order of decreasing task priority. Each TCB in the active chain is connected by its link word to the next TCB in the chain. Among equal priority tasks, a round-robin scheduling of system resources is performed. Whenever a task has its TCB entered in the active chain, the task is automatically assigned the lowest priority within a priority level. The last TCB in the active chain has a link of -1.

Unused TCBs in the system are linked together to form an inactive chain of available TCBs. Except for the link words, these TCBs are unused until a task is created, at which time a TCB is removed from this chain and placed on the active chain.



Task State Transitions

## TASK EXECUTION CONTROL

Unlike .SYSTEM calls, task calls consist of single word instructions. Task calls are used to create tasks, modify their priority, etc. The differences in structure between an RDOS system call and a task call are the following. First, task calls require a one word call with all parameters passed in the accumulators. Secondly, not all task calls have error returns. Those which do not have error returns do not reserve error return locations. Third, task calls are processed in user address space, while system calls require system action. Finally, task calls are not preceded by the .SYSTEM pseudo-op.

Each task call has an associated modular package of code needed to perform the call operation. Thus for each type of task call used, the task call name must be referenced by an .EXTN statement in the user program. This feature enhances core utilization since only those modules selected by the user will occupy program space at load time. Upon return from a task call, program control is routed through the Task Scheduler to the highest priority task that is ready.

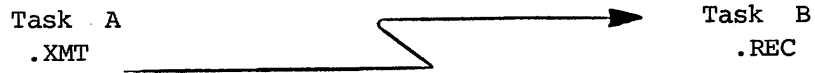
Each task call causes a task state transition. Most task calls cause the issuing task to move from the executing state to the ready state. Task calls in this category are the following:

- .TASK - Create a task.
- .PRI - Change a task priority.
- .ARDY - Make an entire class of tasks ready.
- .ASUSP - Suspend an entire class of tasks (of a different priority from caller).
- .XMT - Transmit a task message.
- .XMTW - Transmit a task message and wait (but no wait required).
- .IXMT - Transmit a task message from an interrupt service routine.
- .REC - Receive a task message (which has already been sent).
- .TOVLD - Perform a multitask overlay load (of a currently resident overlay).
- .OVREL - Release a multitask overlay.
- .OVKIL - Kill an overlay task.
- .QTSK - Queue an overlay task.

The executing task is suspended whenever it issues a system call, in some cases when it issues task calls .ASUSP, .XMTW, .REC, or .TOVLD, and whenever it issues the task call .SUSP which causes the executing task to suspend itself.

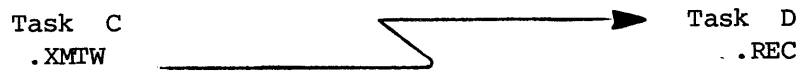
Issuance of task calls can result in a suspended task becoming readied. Calls in this category are .ARDY, .IXMT, .REC, and .XMT .

When the executing task issues a .TASK command, not only is the executing task raised to the ready state but the task which has been activated is raised from the dormant state to the ready state. Conversely, whenever the task issues a .KILL call, the task goes from executing to dormant. .AKILL reduces all tasks of the specified priority level to the dormant state including the caller if the caller's priority is the same as the priority specified in the call.



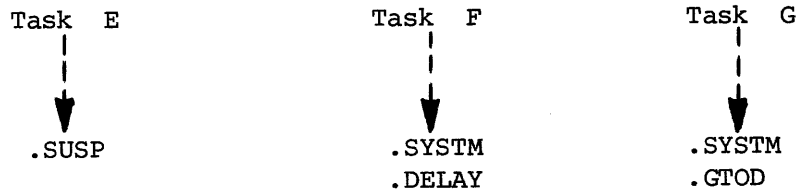
Task A sends message and goes into the ready state.  
Task B is suspended until the message is received.

### Intertask Communication



Neither task proceeds until message is passed from  
Task C to Task D.

### Task Synchronization



### Task Timing Control

## INTERTASK COMMUNICATION/SYNCHRONIZATION

Even though tasks operate asynchronously, it is often desirable for one task to be capable of talking to another task. Tasks communicate with one another under RDOS by sending and receiving one word messages in agreed-upon core locations. One word messages may, of course, be pointers to larger messages if the tasks agree beforehand on the use of such a technique.

A transmitting task may simply deposit the message in an agreed-upon location (by means of the `.XMT` call), or the caller may deposit the message and wait until its receipt (`.XMTW`). To receive such a message another task issues a `.REC` task call. If the transmitting task has not yet sent the message when the `.REC` call is issued, the receiving task waits until the message is sent. If the message has already been sent, the receiving task accepts the message and is put into the ready state for execution when it becomes the highest priority task. If the message was sent via a `.XMTW` task call, both the receiving task and the transmitting task are put into the ready state before control is sent to the task scheduler. If more than one task has issued a `.REC` using the same core address for communication, they will all be put into the ready state by a transmitting task issuing either a `.XMT` or `.XMTW`.

It is possible too for a message to be sent to a receiving task from a user interrupt service routine; this is done by means of the `.IXMT` call.

## TASK TIMING CONTROL

A task may suspend itself for a period of time which the task specifies. This allows users to implement a time slicing or round-robin allocation of CPU control to users. The delaying of a task is accomplished by means of a system call `.DELAY`. Tasks may suspend themselves for time periods that are multiples of the real time clock cycles.

RDOS also maintains a system clock and calendar for those tasks that should be scheduled on a time-of-day basis. Tasks may examine the frequency of the system clock and may obtain or set the date or the correct time in seconds, minutes, and hours. Moreover, tasks within user overlays may be scheduled to gain control at periodic intervals by means of a call to `.QTSK`.

Finally, RDOS provides a pair of system commands which permit the definition and removal of a user clock driven by the system clock. This user clock generates interrupts at user-definable intervals. When one of these interrupts occurs, the Scheduler and task environment are frozen and control goes to a user-specified routine outside the multitask environment.



```

C      MAIN LINE PROGRAM
      CHANTASK      1,3
      EXTERNAL     PROGA, PROGB
C      CREATE TASK-A AT PRIORITY 100
      CALL      FTASK (PROGA,$10,100)
C      CREATE TASK-B AT PRIORITY 100
      CALL      FTASK (PROGB,$10,100)
C      TERMINATE MAIN LINE PROGRAM
      CALL      KILL
C      ERROR RETURN FROM TASK-CREATE CALL
10     WRITE ( 10) "NOT ENOUGH TCB'S"
      END

```

```

C      TASK TO OUTPUT "MSG1" EVERY 30 SECONDS
      TASK      PROGA
10     CALL      FDELY      (10*30)
      TYPE      "MSG1"
      GO TO 10
      END

```

```

C      TASK TO OUTPUT "MSG2" EVERY 45 SECONDS
      TASK      PROGB
20     CALL      FDELY      (10*45)
      TYPE      "MSG2"
      GO TO 20
      END

```

		{ MSG1 MSG2 MSG1 MSG2 MSG1 MSG1 MSG2 MSG1 MSG2 MSG1         }
Program	Output	

## REAL TIME FORTRAN

Data General's Real Time (RT) multitask FORTRAN is a superset of single task DGC FORTRAN IV. Each assembly language task command discussed earlier has a counterpart in RT FORTRAN. Each RT FORTRAN task command has been implemented as a FORTRAN CALL statement. This means that a user can code his multi-tasking application in a high level language like FORTRAN with which he is probably more familiar. The following list describes briefly each of these calls:

CALL FTASK (NAME, ERROR, PRIORITY)	Initiate a task
CALL AKILL (PRIORITY)	Kill a class of tasks
CALL KILL	Kill the calling task
CALL ASUSP (PRIORITY)	Suspend a class of tasks
CALL SUSP	Suspend the calling task
CALL ARDY (PRIORITY)	Ready a class of tasks
CALL PRI (PRIORITY)	Change the calling task's priority
CALL XMT (KEY, MESSAGE, ERROR)	Transmit a task message
CALL XMTW (KEY, MESSAGE, ERROR)	Transmit a message and wait
CALL REC (KEY, MESSAGE)	Receive a task message
CALL FDELY (TICKS)	Delay the calling task
CALL FSTIM (HR, MIN, SEC, DAY)	Set the time of day and date
CALL FGTIM (HR, MIN, SEC, DAY)	Get the time of day and date

Two FORTRAN specification statements are used to name a user task module and to specify the maximum number of tasks and channels which will be active at any single moment. These statements (CHANTASK and TASK) are shown in the illustration program on the previous page.

The illustration program given as an example consists of three FORTRAN modules, MAIN, PROGA, and PROGB. The purpose of this program is to perform two asynchronous tasks: Print a message ("MSG1") on the Teletype every 30 seconds and a second message ("MSG2") on the Teletype every 45 seconds.

The CHANTASK statement in the main line program specifies that a maximum of 3 tasks will be active at any one time and that one channel will be required by the program. The main line program activates PROGA and PROGB, after which it KILLS itself. An error message output is provided ("NOT ENOUGH TCB'S") for the error return required by FTASK.

LIST OF COMMAND WORDS

.APPEND Open a file for appending.  
.BREAK Save the current state of memory in save file format.  
.CCONT Create a contiguous file.  
.CHATR Change file attributes.  
.CLOSE Close a file.  
.CRAND Create a random file.  
.CREATE Create a sequential file.  
.DELAY Delay the execution of a task.  
.DELET Delete a file.  
.DIR Change the current default directory device.  
.ERTN Return with exceptional status.  
.EXEC Execute a save file program swap.  
.EXFG Execute a program in the foreground.  
.GCHAR Read a character from the console TTI.  
.GDAY Get the current date.  
.GTATR Get file or device attributes.  
.GTOD Get the current time.  
.IDEF Identify a user interrupt.  
.INIT Initialize a mag tape or directory device.  
.INST Install a new RDOS system from the default directory device.  
.IRMV Remove a user interrupt.  
.MEM Determine available memory space.  
.MEMI Allocate an increment of memory.  
.OPEN Open a file.  
.OVKIL Kill a user overlay task.  
.OVLOD Load a user overlay.  
.OVOPN Open user overlays.  
.PCHAR Write a character to the console TTO.  
.QTSK Queue an overlay task.  
.RDB Read in block mode.  
.RDL Read a line.  
.RDR Read a random record.  
.RDS Read sequential.  
.RENAM Rename a file.  
.RESET Close all open files.  
.RLSE Release a directory device, preventing further access.  
.RTN Return to the previous program swap.  
.SCALL 0 Get the Real Time Clock frequency.  
.SCALL 1 Define a user clock.  
.SCALL 2 Turn off a user clock.  
.SDAY Set the system calendar.  
.SPDA Disable device spooling.  
.SPEA Enable device spooling.  
.SPKL Stop a spooling operation.  
.STOD Set the time of day.  
.WRB Write in block mode.  
.WRL Write a line.  
.WRR Write a random record.  
.WRS Write sequential.

## SYSTEM CALLS

Users communicate with the Real Time Disk Operating System by making a system call followed by a system command word. The general form of a system call is as follows:

```
.SYSTEM  
command  
exceptional return  
normal return
```

The mnemonic `.SYSTEM` (defined by the RDOS assembler to be `JSR @17`) must precede each command word. It passes control to the system through the task monitor. The task monitor saves the task environment in the calling task's TCB before passing control to RDOS. A list of legal system command word mnemonics is given on the opposite page.

There are three basic command word formats:

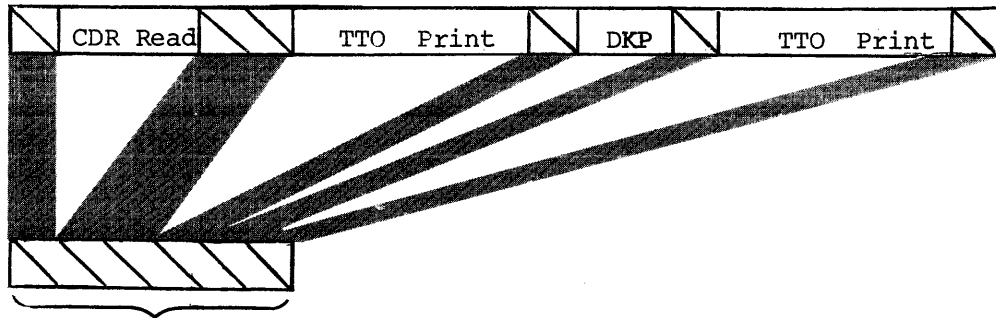
1. `command`
2. `command n`
3. `.SCALL p`

In the first command word format, the command is a predefined mnemonic (as shown on the previous page) which is not followed by an integer.

In the second format, n is a positive integer representing an I/O channel number. The channel number n indicates a logical link to an opened file or device. The largest value of n may not exceed 768. When no I/O channel is needed in the command execution, the command word appears alone in the instruction. If the command requires arguments, these are passed in the accumulators.

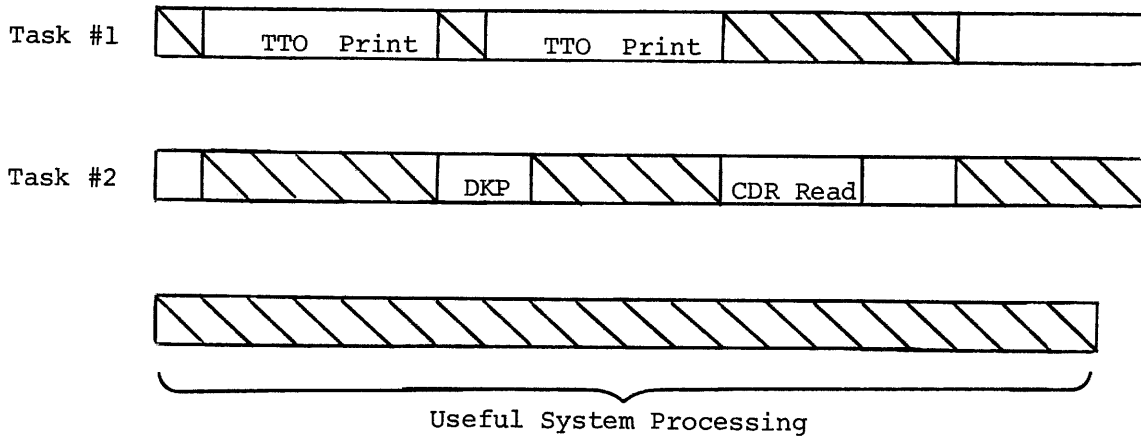
Any system command requiring a channel number n need not specify this number in the command itself. By specifying octal 77 as the channel number in the instruction, the system will use instead the number passed in AC2. This gives the user a very flexible runtime device selection method.

In the third command word format, the mnemonic `.SCALL` is followed by a digit p to specify a particular command type.

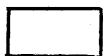


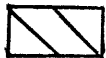
Useful System Processing

Single Task Operating System



Multiple Task Operating System

 I/O Processing or Task Suspension

 Useful System Processing

## RDOS INPUT-OUTPUT CONTROL

An important function of any real time operating system is the efficient handling of input-output operations. Optimum usage of machine devices and central processor time in the accomplishment of tasks is the real reason for designing and implementing a multi-tasking system.

Since I/O devices are slow compared to the internal speed of the computer, they must be programmed to overlap their operations with computations, when possible, in order to:

- . Increase usable CPU time
- . Greatly increase efficiency of I/O operations
- . Provide more throughput of data

The responsibility of RDOS I/O control is to react during normal program execution to the structuring of I/O requests, making assignments of requests to machine devices when they are idle, and queuing requests for devices which are busy. Through the queuing facility, RDOS makes it possible to achieve maximum and continuous overlap of multi-tasks without direct intervention by the tasks themselves.

All input and output of data via devices permanently installed in the system must be done via system I/O commands. The system does not reject any user I/O commands, but the issuance of any such commands by a user would be both risky and unnecessary since a full complement of system I/O commands is provided.

As described earlier, system I/O commands require a channel number (0-76) to be given in the second field of the command word. This channel number is assigned to a particular device or file when the device or file is first opened. Devices are opened by means of the system command .OPEN which associates a given file name with a channel number. Having made this association, all commands pertaining to the file merely require that file's channel number.

I/O Modes

<u>Type</u>	<u>Call</u>	<u>Data</u>	<u>Termination</u>
Line	.SYSTEM .RDL (.WRL) n*	ASCII (even parity)	carriage return form feed null 132 characters
Sequential	.SYSTEM .RDS (.WRS) n*	binary	byte count = 0
Random Record Access	.SYSTEM .RDR (.WRR) n*	binary	64 word record
Direct Block	.SYSTEM .RDB (.WRB) n*	binary	m logical 256 word blocks

\*n is the logical channel number associated with the desired device or file name.

## INPUT/OUTPUT COMMAND MODES

The system provides four basic modes for reading and writing data: Line, Sequential, Random Record Access, and Direct Block.

In Line Mode, data read or written is assumed to consist of ASCII character strings terminated by carriage returns, form feeds, or nulls. Reading or writing continues until one of these three characters is detected. The system handles all device-dependent editing at the device driver level. For example, line feeds are ignored on paper tape input devices and are supplied after carriage returns on all paper tape output devices. Furthermore, neither reading nor writing require byte counts, since reading continues until a terminator is detected and writing proceeds until a terminator is written.

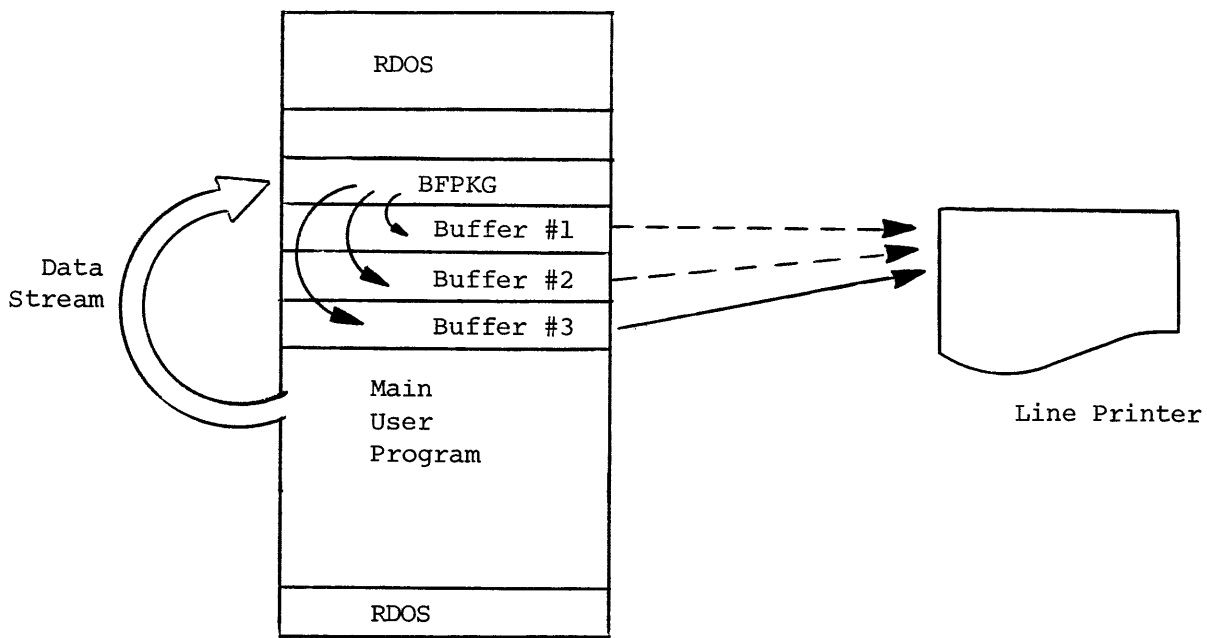
Sequential mode provides unedited data transfers. In this mode, no assumption is made by the system as to the nature of the information. Thus this mode would always be used for processing binary data and could also be used for processing ASCII data (provided no editing of this data is required). Sequential mode transfers require specific byte counts in order to satisfy read or write requests. All I/O devices may be used in sequential data transfers.

Random Record Access mode permits 64-word segments of disk block data to be accessed randomly. Data which is transferred in this mode may be either ASCII or binary, although no device level editing occurs.

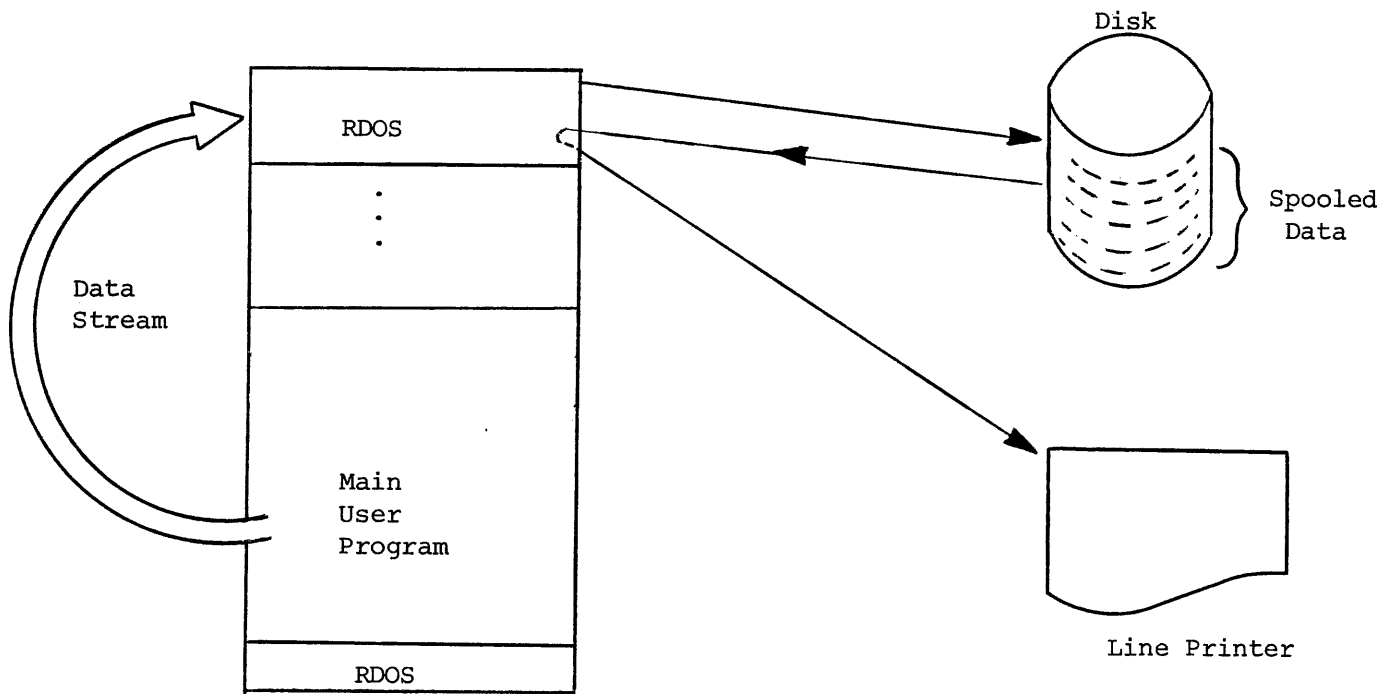
In Direct Block mode, binary data is transferred directly between a disk file and a specified core area. This specified core area takes the place of the system buffer which is required for all other types of data transfers. The elimination of a system buffer makes direct block transfers faster than all other types of data transfers.

Two additional system commands are provided which permit unedited character transfers via the console teletypewriter: .GCHAR and .PCHAR . These two commands do not constitute an I/O mode since only the console teletypewriter is referenced.





Buffered I/O Package, BFPKG



Simultaneous Peripheral Operation On Line (SPOOLING)

## SYSTEM INPUT/OUTPUT BUFFERING

All data transfers (except Direct Block) to or from disk files and hardware devices are buffered in the operating system before the data is transferred into the user's buffer. Each system device handler has a small buffer associated with it depending on the speed of the device, e.g., 162 (decimal) words for the card reader, 40 bytes for the paper tape punch, and 160 bytes for the line printer. The other area is the system buffer area which is organized into blocks of 256 words each. When transferring data from a disk file it is first read into this buffer area before accessing the data within the block. This allows smaller transfers of data to or from the user area.

## SPOOLING

Efficient I/O handling is the most important single factor in the effective utilization of CPU time. When messages are output on a slow device like a Teletype and its buffer fills up, the calling task will be suspended until the buffer is emptied. When spooling is provided, the next message called is temporarily stored on disk, and is later returned to core when the current message is completed. The significance of spooling is that queuing of output messages or information can now be accomplished easier without putting excessive loads on user core partitions. This also frees the user from having to optimize his message requests, thus permitting more effective use of the device. Spooling normally operates transparently on the Teletype, paper tape punch, plotter, and line printer but system and CLI commands allow the user to control spooling.

## BUFFER CONTROL PACKAGE

The RDOS system library provides a module which permits faster line and sequential I/O transfers than is possible using the system I/O calls discussed previously. It utilizes tasking concepts to fill two or more buffers asynchronously and therefore provides a constant supply of data for program processing.

SYS.DR    File    Entry

<u>Displacement</u>	<u>Contents</u>
0-4	File name in ASCII, bytes packed left/right, left justified, trailing null bytes.
5	Two-character name extension (SV, RB, etc.); trailing null bytes.
6	File attributes (permanent, etc.).
7	File Block Count, less one.
10	Byte count in last block.
11	Address of the first block in the file.
12	Device code.

SYS.DR    Device    Entry

<u>Displacement</u>	<u>Contents</u>
0-4	Device name in ASCII, bytes packed left/right, left justified, trailing null bytes (\$PTR, etc).
5	0
6	Device attributes (permanent, attribute-protected, and read/write protected).
7	0
10	0
11	0
12	Device code.

## DISK FILE ORGANIZATION

The term "file" applies to any collection of information contained in blocks of disk storage. Typical examples include a source program file, a relocatable binary file, and a core image (save) file.

All files and devices are accessible by file name. A basic file name is a string of alphanumeric characters and the character \$. A file name can contain any number of characters, but the system considers only the first 10 significant. A file name extension can be appended to a file name as a qualifier. Such an extension is a string of alphanumeric characters and may include the character \$; the system considers only the first two significant. A period (.) separates the extension from the file name. When the character \$ is the first character of a system file name, a hardware device is indicated.

Some typical file names that might be found on a disk pack are as follows:

ABC.RB	<u>r</u> elocatable <u>b</u> inary
SYS.SV	core image ( <u>s</u> ave file)
CLI.OL	<u>o</u> verlay file
\$LPT	line printer

All files have attributes which characterize them. These file attributes can be set and changed by the user, and include such items as:

C	-	<u>c</u> ontiguously organized file
D	-	randomly organized file
P	-	<u>p</u> ermanent file, one which cannot be deleted or renamed
S	-	<u>s</u> ave file (core image)
W	-	<u>w</u> rite-protected file, one which cannot be written

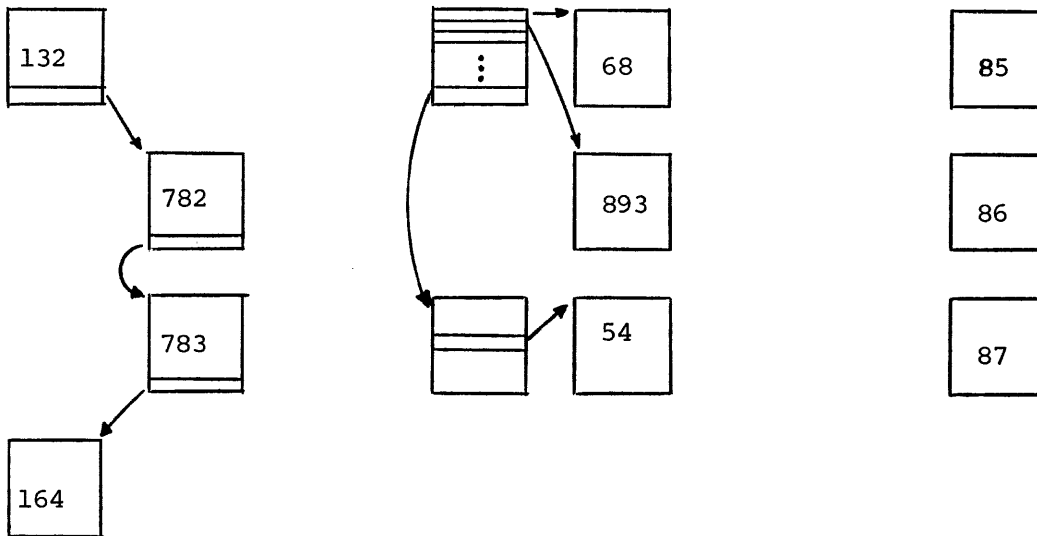
Names of all files on any moving head disk surface are contained in a table called the system directory, SYS.DR. (The directory itself is a disk file.) SYS.DR contains the file name, attributes, and all other pertinent information describing each file on the disk. Another table is used to keep track of the availability of each block for data storage. This table is called the map directory, MAP.DR, and it too is a disk file. The system scans MAP.DR whenever a disk file is to be written so that those blocks which are free can be found and used. Whenever a file is deleted, the disk space formerly occupied by the file becomes free for other storage, and that file's entry in SYS.DR is removed.

TYPE: Sequential Random Contiguous

CREATION: .SYSTEM .SYSTEM .SYSTEM  
 .CREATE .CRAND .CCONT

EXTENDIBILITY: yes yes no

ORGANIZATION:



ACCESS COMMANDS: RDL/WRL RDL/WRL RDL/WRL  
 RDS/WRS RDS/WRS RDS/WRS  
 RDR/WRR RDR/WRR RDR/WRR  
 RDB/WRB RDB/WRB

MAXIMUM REQUIRED DISK SEEKS: n (where n is the number of blocks in the file) 2 1

Disk File Structures

## DISK FILE STRUCTURES

### Sequentially Organized Files

Sequential organization is the simplest organization to understand. Information in sequentially organized files is stored in groups of disk blocks. The last word of each 256 word block is used to store a pointer to the next block in the file. This pointer is invisible to the user, and is solely for system use. Each 256 word block has a unique address called a physical block address which is derived from the physical disk sector/track addresses. Distinct from the physical block addresses is the logical block number which denotes the relative position of a block of data within its disk file.

The physical block addresses of a sequentially organized file need not be (and seldom are) in an unbroken series. When building a sequential file the system simply appropriates the next available disk block when storage is needed, and constructs a pointer to the block. Sequentially ordered blocks are sequential in this sense: After processing any given block, the system may step either to the previous block or to the next block in the series. To access the tenth block after the first block, the system would have to read the nine intervening blocks -- a time-consuming process.

### Randomly Organized Files

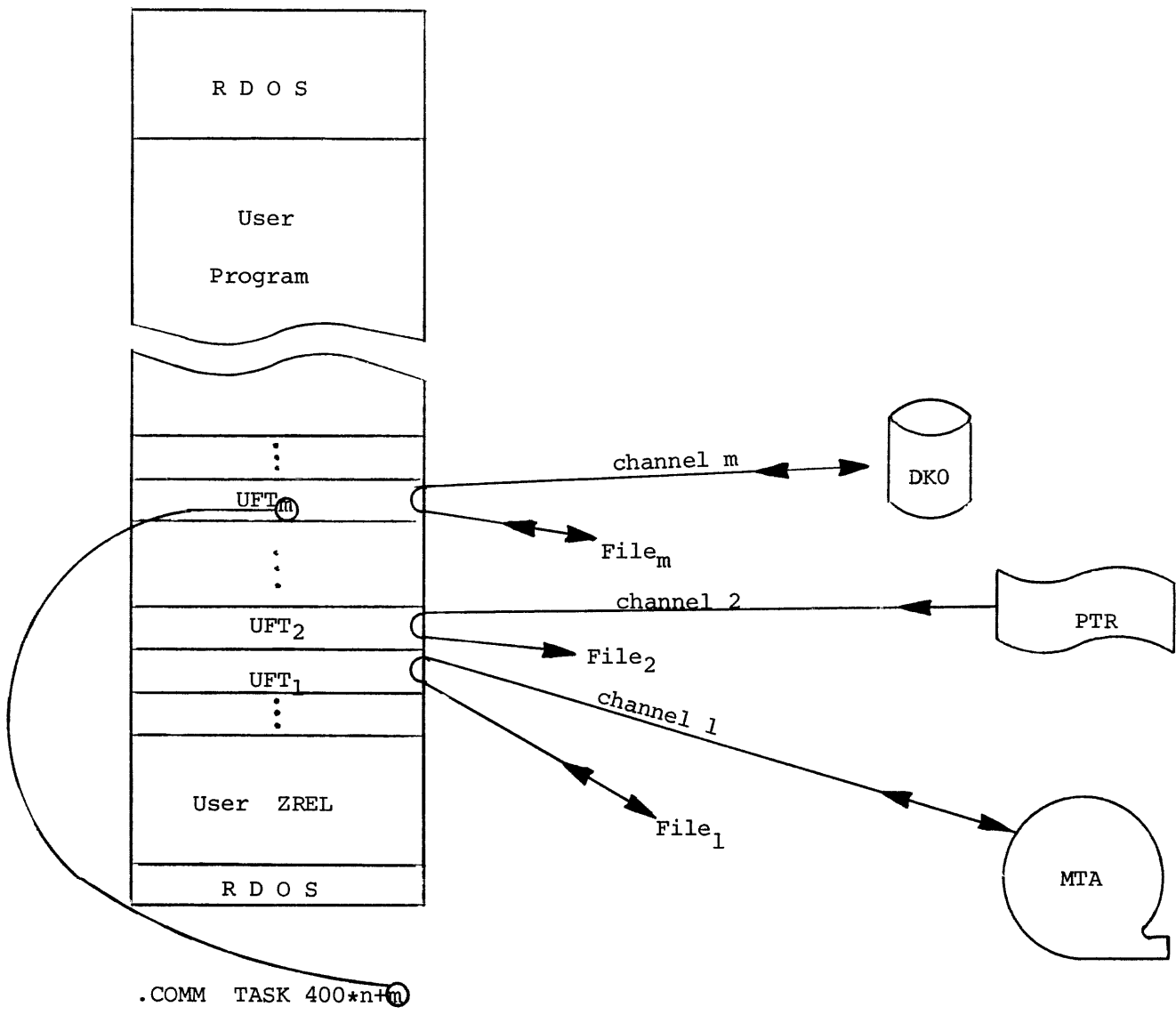
Random file organization provides the best combination of flexibility and accessibility of data. In randomly organized files a master index of all physical block addresses is created. The master index blocks themselves are sequential files.

Blocks of data storage in random files utilize all 256 words for information storage. Each block is assigned a sequential positive integer by its position within the master index, indicating the block's logical position within the file. In processing randomly organized files, two disk accesses at most are all that is generally required for the reading or writing of each block: One to access the file index and one for the block of data itself. If the index is core resident (having previously been read into a system buffer), only one access need be made. In most cases the index will be core resident.

### Contiguously Organized Files

Contiguous file organization has a rigid structure yet provides the quickest access to data. Contiguous files are composed of a fixed number of disk blocks which constitute an unbroken series of disk block addresses. These files can neither be expanded nor reduced in size, since by definition they occupy a fixed series of disk blocks. Contiguous files may be considered as files whose blocks may be accessed randomly but without the need for a random file index.

All I/O operations which can be performed on randomly organized files can be performed on contiguously organized files. Contiguous files have the advantage of usually requiring less time for accessing blocks within the file. The draw-back to contiguous files is that they may not always be created, and may never be changed in size. Their creation depends upon the availability of the required number of free neighboring disk blocks.



Disk File Input/Output Control

## DISK FILE INPUT/OUTPUT CONTROL

Since there are commonly more files and devices than there are communication channels, a series of tables are maintained to specify which files or devices are currently associated with each channel. These tables are core resident and are called User File Tables (UFTs). There is one UFT for each channel specified in the user program.

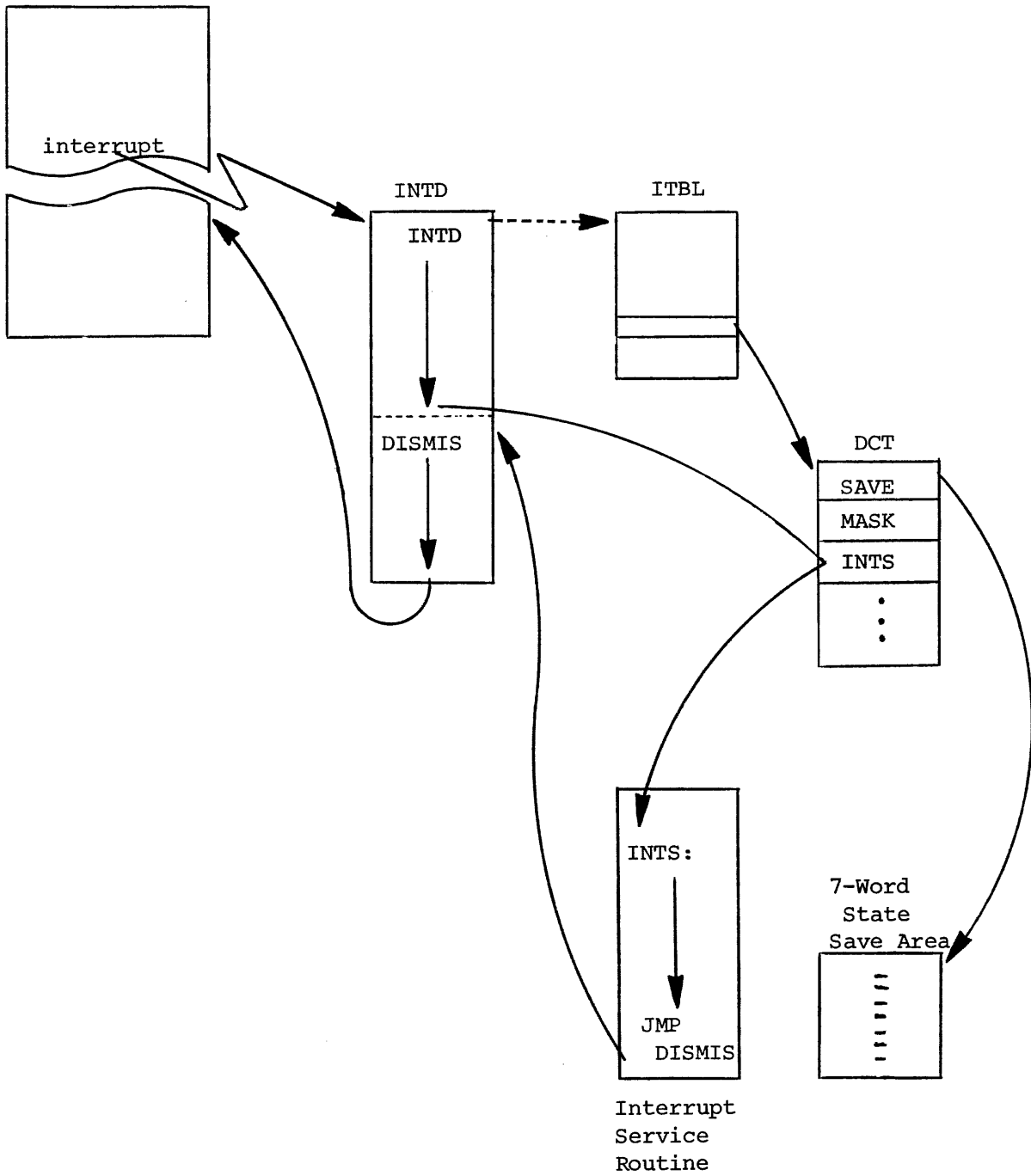
When a file is opened on a specified system channel, that file's description is copied from its System Directory entry into the UFT. This information is required for processing the file and includes the file name and its extension, its attributes, and file access information.

All UFTs are built in contiguous frames. The first UFT is associated with channel 0, the second with channel 1, etc. up to the maximum specified by the user at program load time. Immediately preceding the UFTs is a small table called the User File Pointer Table (UFPT) which contains pointers to each UFT. The system uses the channel number to index the UFPT, getting the UFT address whenever a system command references a channel number.

In summary, to access a file or device the user first opens this file on a currently unused channel, setting up a UFT for this file. Each UFT contains all the file and I/O information required to associate a given file and channel at open-time. The UFT also contains information on the current status of any active I/O requests whether they are sequential (.WRS or .RDS), line (.WRL or .RDL), random record (.WRR or .RDR), or direct block (.WRB or .RDB) calls.



User Program



Flow of Control During Interrupts

## INTERRUPT SERVICING PROGRAM

When an interrupt is detected by the hardware, the currently executing program is suspended and control goes to an interrupt dispatch program, INTD. INTD is an integral portion of the RDOS system and resides in core at all times. It directs control to the correct servicing routine by using the device code as an index into an interrupt branch table. The entry in this table is the address of a device control table (DCT) associated with the servicing routine.

The first three entries of the DCT are as follows:

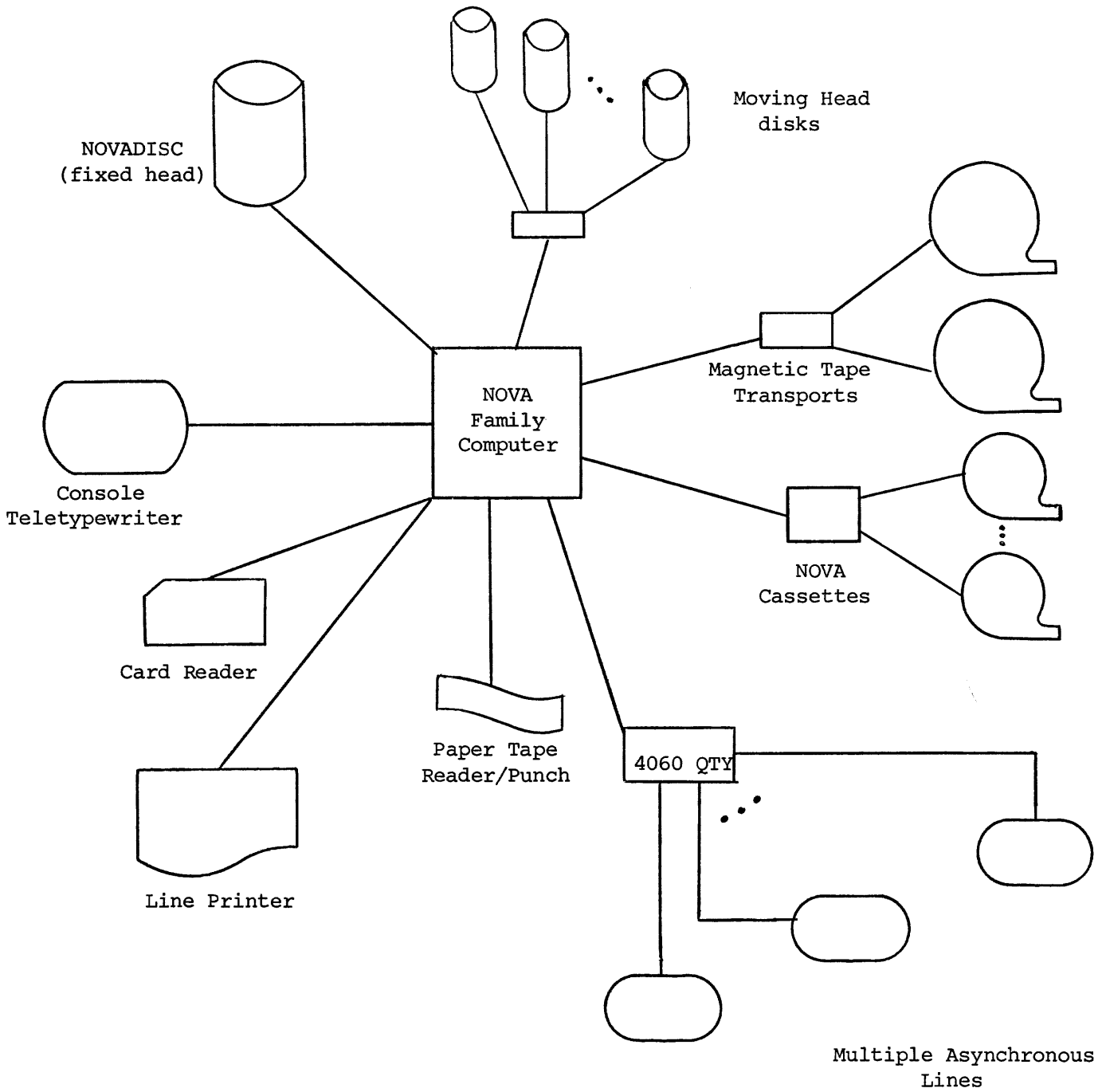
<u>Word</u>	<u>Menmonic</u>	<u>Purpose</u>
0	DCTSV	Address of 7 word state save area
1	DCTMS	Interrupt service mask
2	DCTIS	Device Interrupt Service Routine Address

## USER INTERRUPT PROCESSING

Enough said about standard I/O devices. What about non-standard devices, the type customers are always most interested in? For these devices, a simple software interface exists to attach non-standard user devices to RDOS. This interface is provided through an abbreviated DCT (the first three entries of a standard DCT) which supplies the address of a 7-word state save area, the hardware interrupt mask to be set while servicing the user interrupt, and the address of the interrupt servicing routine. The system will store the program counter, accumulators, carry, current hardware mask, etc. in the state save area before transferring control to the interrupt service routine. The interrupt service routine is written by the user, and contains all program code necessary for processing the interrupt.

A system call, `.IDEF`, is used to insert pointers to user DCTs into the interrupt vector table, identifying the device to the system. Input parameters necessary for the call to `IDEF` are the device code of the user device and the DCT address of the user written driver. To remove these entries from the table, the system call, `.IRMV`, is issued with the device code passed as a parameter.

It is possible to activate user tasks from the interrupt servicing world. This is done by transferring a non-zero message from the interrupt servicing routine to a user task via the `.IXMT` task monitor call. If the task expecting such a message has issued a `.REC` call, the task will be put into the ready state by the `.IXMT` call. If `.REC` has not been issued, the `.IXMT` call simply posts the message and the interrupt service routine finishes its processing.



RDOS Hardware Configurations

## MULTIPLE DEVICES AND UNITS

The Real Time Disk Operating System is capable of supporting multiple disk and tape storage units, and multiple data processing devices.

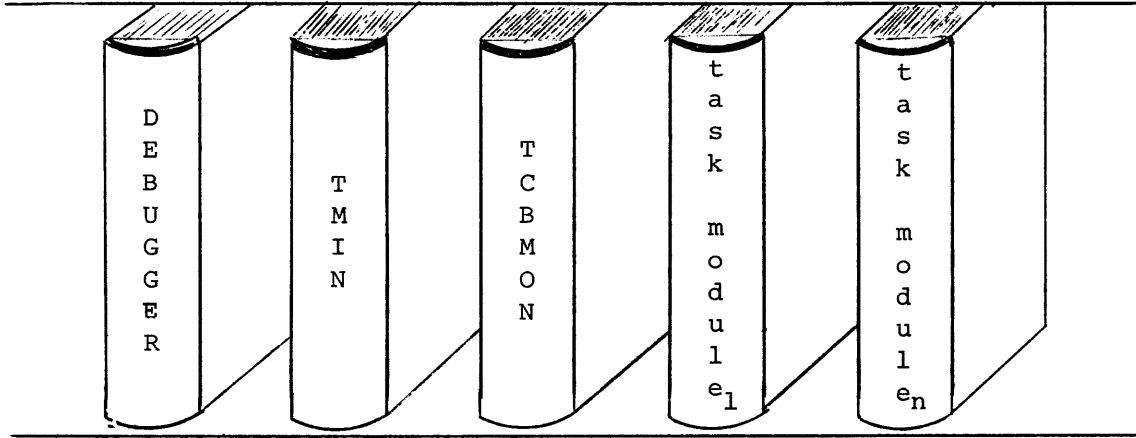
The master system device for RDOS can be either a fixed or a moving head disk. RDOS supports a fixed head NOVADISC with total storage from 128 to 2048 thousand words. Up to four moving head units (disk pack or cartridge type) can also be included in any system, with from 2 to 20 disk surfaces per unit; maximum total storage is 49.2 million words. Disk files are referenced by specifying both the disk unit number and the file name, since every disk unit contains its own system directory. Referencing a file named TEST on a fixed head disk via the CLI is accomplished by means of the expression DK0:TEST .

Referencing a file named TEST on a disk pack (unit 2, for example) via the CLI is done by means of the expression DP2:TEST. Prefixing the file name by a device specifier is only necessary when the device is not the current directory device.

From 1 to 8 magnetic tape transports and from 1 to 8 cassette drives can also be supported by RDOS. Magnetic tape units must be set at high density (800 bpi), and both 7 and 9 track tapes are allowed. Individual files can be referenced on both magnetic tape and cassette units by specifying both the transport drive and file number. Up to 100 files can be referenced on each unit. To reference a file via the CLI or from a user program, the file must be specified by a file number as part of the file name. An example of a CLI command to reference the second file on magnetic tape transport 7 is XFER MT7:1 TEST. This command would load the second file on transport 7 and transfer it to disk under the file name TEST.

The type 4060 asynchronous data communications multiplexor (QTY) is another device which can be considered to be unit expandable. The QTY can accommodate from 1 to 64 full duplex lines at speeds up to 9600 baud. Each multiplexed line of the QTY corresponds to a file name of the form QTY:x, where x is the multiplexor line number in the range 0 to 63. Input/output operations are performed on each line by RDOS line/sequential reads and/or writes.

Since the system device drivers have been written reentrantly, multiple devices can be easily supported by RDOS. This can be accomplished by simply adding another device control table (DCT) and by using existing device driver routines. Devices in this category are the teletypewriter, paper tape reader/punch, line printer, card reader, and incremental plotter.



RDOS System Library

TMIN - single task monitor  
TCBMON - multitask monitor, .TASK and .KILL logic  
Task Module<sub>1</sub> - .AKILL, .ASUSP, .ARDY logic  
Task Module<sub>n</sub> - other task processing logic

## SYSTEM LIBRARY

The system library (SYS. LB) is a collection of program modules which support user programs run under RDOS. These modules can be likened to volumes on a library shelf. Each user program needing one or more of these modules selects them from the library by means of the Relocatable Loader, leaving behind those modules which are of no current use. By placing system modules in the library, a savings in user program core storage requirements thus results.

Whereas the DOS system library contained only the debugger, the RDOS system library also contains the multi and single task schedulers (TMAX and TMIN), command processing modules for each task call type, and the Buffered I/O package (BFPKG) which has been discussed earlier.

Modules are extracted from this library by the relocatable loader. This loader is told which modules to load either by switches in the relocatable load CLI command or by means of the external normal pseudo-op, .EXTN . Except for the loading of a Task Scheduler, only those task modules will be loaded which are referenced by an .EXTN statement in the user program. The program must externally reference each task call name that is issued as a task call by the user program, or the loader will be unable to resolve the call. By only loading those task modules which are required for program operation a net savings in total core requirements is obtained. All modules taken from the library are loaded after the main (or root) program. Thus the loading of user programs and library modules occurs from low core to high core.

In addition to the debugger, task schedulers and other task modules, the Buffered I/O package (BFPKG) is also found. This module further enhances system operation by providing the user with asynchronous Line and Sequential data transfers, buffered in user program space.

- . Binary Loader
- . Compile, Load, and Go
- . Command Line Interpreter
- . Symbolic Debugger
- . Extended ALGOL
- . Extended Real-Time FORTRAN IV
- . Library File Editor
- . Octal Editor
- . Extended Assembler
- . Macro Assembler
- . Background Text Editor
- . Foreground Text Editor
- . Relocatable Loader

Real Time Disk Operating System Programs

## RDOS SUPPORTED SOFTWARE

The Real Time Disk Operating System (RDOS) can be used for both the development and the implementation of user programs in both real-time and non real-time applications. As discussed earlier RDOS includes all the file capabilities normally only available on large machine disk operating systems, allowing the user to edit, assemble, execute, debug and compile, and create, extend and delete files.

Software currently supported under RDOS includes:

1. Text Editor, for editing and updating program files.
2. A foreground Text Editor.
3. Extended Assembler producing relocatable or absolute binary output from symbolic source programs.
4. Relocatable Loader for linking relocatable binary files into an output core image file.
5. Extended FORTRAN IV with language extensions.
6. Extended ALGOL compiler which provides many facilities in addition to those of ALGOL 60.
7. Library File Editor enabling the user to easily separate, edit, and maintain relocatable binary program libraries.
8. Debug III which allows symbolic debugging of user programs.
9. Octal Editor enabling the user to examine and modify the contents of disk files.



CHANGES FROM REVISION 1 TO REVISION 2 OF THE INTRODUCTION TO THE REAL TIME DISK OPERATING SYSTEM

Substantive changes and additions to this manual are described on the following page by page basis. Typographical corrections are not included in this list.

- |                              |   |
|------------------------------|---|
| Pages 6 and 7                | These new pages describe foreground/background programming concepts as found in RDOS revision 1 .                           |
| Pages 8 and 9                | These new pages describe a memory protection and mapping device which has been developed for the NOVA 800 series computers. |
| Pages 10 and 11              | These new pages describe disk partitioning concepts as applied to single and dual processor systems.                        |
| Page 13                      | Page 13 has been expanded to describe the use of keyboard interrupts for communicating with RDOS .                          |
| Pages 14, 26, 27, 29, and 32 | These pages have been expanded to include additional system and CLI commands made available by RDOS revision 1 .            |
| Page 18                      | This illustration has been expanded to depict multiple overlay areas.   |

DATA GENERAL CORPORATION  
PROGRAMMING DOCUMENTATION  
REMARKS FORM

DOCUMENT TITLE \_\_\_\_\_

DOCUMENT NUMBER (lower righthand corner of title page) \_\_\_\_\_

Specific Comments. List specific comments. Reference page numbers when applicable. Label each comment as an addition, deletion, change or error if applicable.

cut along dotted line

General Comments and Suggestions for Improvement of the Publication.

FROM: Name: \_\_\_\_\_ Date: \_\_\_\_\_  
Title: \_\_\_\_\_  
Company: \_\_\_\_\_  
Address: \_\_\_\_\_  
\_\_\_\_\_

FOLD DOWN

FIRST

FOLD DOWN

FIRST  
CLASS  
PERMIT  
No 26  
Southboro  
Mass 01772

**BUSINESS REPLY MAIL**

No Postage Necessary if Mailed In The United States

Postage will be paid by:

**Data General Corporation**

Southboro, Massachusetts 01772

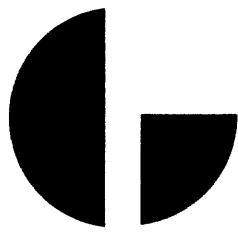
**ATTENTION: Programming Documentation**

FOLD UP

SECOND

FOLD UP

STAPLE



**DATA GENERAL  
CORPORATION**

Southboro,  
Massachusetts 01772  
(617) 485-9100